Tampere University

Viljami Männikkö

SPLINE OPTIMIZATION

Spline optimization using genetic algorithm in AGV system

Facility of Engineering and Natural Sciences
Master of Science Thesis
October 2019

ABSTRACT

Viljami Männikkö: Spline optimization
Master of Science Thesis
Tampere University
Master's Degree Programme in Engineering and Natural Sciences
October 2019

Splines are polynomial curves. They are used for example to model complicated shapes in computer programs and science. They can be also used to construct planes. Those planes can be used to easily model complicated surfaces. Splines are widely used also in 3D modeling, where models surfaces are defined with splines. In this work, we use splines to model **automated guided vehicles' (AGV)** routes. By combining splines, we can easily construct complex route networks to all sorts of vehicles.

Splines can be constructed in multiple ways. No matter of construction method, splines' shape is always defined by using control points. In this work, splines are constructed by using knot vector and so-called basis functions. If splines are constructed in this way, they are called **basis splines (B-splines)**. Any spline can be defined by a linear combination of B-splines and that's why B-splines are practically some sort of basis for splines. B-spline basis functions are constructed by using the Cox de Boor recursion algorithm.

This work's purpose is to study spline properties from a spline modification point of view. Our goal is to develop an algorithm that modifies the spline route to be faster to drive with an automated guided vehicle in a specified system. Optimization is made by using genetic algorithm properties. In route optimization, we need to take into account different kinds of systems' restrictions. Splines model AGV routes in the warehouse, which adds more restrictions to the optimization. In optimization, we need to take into account at least AGV maximum acceleration, maximum deceleration, wheel maximum rotation angle, maximum centripetal and maximum velocity. The route has to also be possible to drive, so we have to check during the route that AGV doesn't hit the obstacles. The goal is to modify route such that AGV start and end velocity, start- and endpoint and start and end angle doesn't change during the optimization. In the work, we will also go through other possible approaches to the optimization, such as the so-called resilient backpropagation method.

Keywords: Spline, Polynomial curve, Genetic algorithm, Optimization, AGV system

TIIVISTELMÄ

Viljami Männikkö: Spline optimointi
Diplomityö
Tampereen yliopisto
Teknis-luonnontieteellinen DI-tutkinto-ohjelma
Lokakuu 2019

Splinet ovat polynomikäyriä, joita käytetään erityisesti mallintamaan monimutkaisia muotoja tietokoneohjelmissa. Splinejen avulla voidaan myös muodostaa tasoja, joiden avulla voidaan mallintaa helposti monimutkaisiakin pinnanmuotoja. Splinejä hyödynnetään myös 3D mallinnuksessa, jossa mallien pinnat määritellään niiden avulla. Tässä työssä niitä käytetään vihivaunujen reittien mallintamiseen. Yhdistelemällä niitä voidaan muodostaa monimutkaisia reittiverkostoja vihivaunuille.

Splinet voidaan konstruktoida erilaisilla tavoilla, mutta kaikille niille on yhteistä se, että niiden muoto määritellään kontrollipisteiden avulla. Tässä työssä splinejen konstruktointiin käytetään solmuvektoria ja niin kutsuttuja kantafunktioita. Tällä tavoin konstruktoituja splinejä kutsutaan **basis splineiksi (B-spline)**. B-splinejen avulla voidaan esittää mikä tahansa spline niiden lineaarikombinaationa, joten B-splinet ovat käytännössä "kantasplinejä". B-splinejen kantafunktioiden konstruktointiin käytetään Cox de Boor:in rekursio menetelmää.

Tämän työn tarkoituksena on tutustua splinejen ominaisuuksiin, niiden muokkauksen näkökulmasta. Tavoitteena on kehittää algoritmi, jonka avulla AGV:eiden reittejä pystytään muokkaamaan nopeammin ajettavaksi. Optimointi suoritetaan hyödyntämällä geneettisten algoritmien ominaisuuksia. Splinejen optimoinnissa tulee ottaa huomioon systeemin erilaiset rajoitteet. Splinet kuvaavat vihivaunujen reittejä varastohallissa, joten tämä luo systeemiin lisää erilaisia rajoitteita. Optimoinnissa pitää ottaa huomioon vihivaunun maksimikiihtyvyys, maksimihidastuvuus, renkaiden maksimikääntökulma, maksimikeskihakuisvoima ja vihivaunun maksiminopeus. Reitin tulee olla myös mahdollinen ajaa, joten reitin aikana pitää myös tarkastaa ettei vihivaunu osu seinään tai toisen vihivaunun reitille. Tavoitteena on siis muokata kulkureittiä siten, että vihivaunun alku- ja loppunopeus, alku- ja loppupiste sekä lähtö- ja tulokulma eivät muutu, mutta kulkureitti kokonaisuudessaan olisi nopeampi ajaa. Työssä käydään myös läpi muita mahdollisia lähestymistapoja optimointiin, kuten esimerkiksi niin sanottu resilient backpropogation menetelmä.

Avainsanat: Spline, Polynomi kurvi, Geneettinen algoritmi, Optimointi, AGV-järjestelmä

PREFACE

This Master of Science thesis was done for the Rocla Oy and work was done at Atostek Oy. Work was guided by the master of science Antti Anttonen from Atostek Oy and professor Esko Turunen from Tampere University. The work was funded by both Atostek Oy and Rocla Oy. Rocla funded the algorithm development part and Atostek Oy funded the writing process and the research to the theory behind the algorithm. I want to thank both of my supervisors, the master of science Olli Kivelä for help with thesis topic creation, Rocla Oy for giving this topic to me and finally Atostek Oy for an opportunity to do this thesis as a part of the project.

Tampere, 18th October 2019

Viljami Männikkö

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

$\mathbf{a}$	Acceleration
AGV	Automated guided vehicle is independently working vehicle.
B-spline	Basis spline is spline, which is constructed by using basis functions and control points.
$\mathbf{F}$	Force
f	Fitness function
$\overline{f}$	Mean fitness of the population
GA	Genetic algorithm, heuristic method for solving the mathematical especially combinator problems.
k	Degree of the spline
m	Mass
M	Number of knots
n	Number of control points
$\mathbb{N}$	Natural numbers $= \{0, 1, 2, ..., \}$
$N_{i,j}$	Basis function
N_b	New population base size
N_c	Number of children in each population
N	Population size
P	Population
$\mathbf{p}_i$	Control point
$p_{better\ favour}$	Better parents favouring probability
p_c	Crossover probability
p_m	Mutation probability
$\mathbb{R}$	Real numbers
RPROP	Resiliant backpropagation
S	Spline is piecewice polynomial curve, which shape is defined using control points.
t	Spline variable
u_i	Knot
x_i	ith candidate of the population

1 INTRODUCTION

In the future, more and more warehouses will be more likely to be automated which means that demand for automated guided vehicles (AGV) is increasing. One solution for the automated warehouse is the automated guided vehicle system offered by Rocla Oy. Rocla is a Finnish electric warehouse truck, AGV and forklifts manufacturer. AGVs are independently working trucks that are controlled by the automated warehouse system. AGVs move along pre-defined routes and they can transport goods from place A to place B independently. [18] To get warehouse automated, we need to create a route network into a warehouse. Route network describes AGV's possible routes in the warehouse. Route network includes all routes to all the possible places of pick-up and drop-off. [18] [1] At this moment, route networks are constructed by hand using computer software. The goal is to automate the route network creation process, due to the increasing demand for automated warehouses.

In the system, the route network's routes are described by using splines, which are polynomial curves. Spline's shape is defined by using control points [20]. In the current automated route network creation development, it is managed to create automatically a rough route network into the warehouse, but it doesn't take into account AGV's drive times or route's optimality. This work's purpose is to develop an algorithm for route optimization from the drive time point of view, which can be later applied to that software.

In optimization, we need to take into account multiple restrictions. System restrictions that we have to at least take account are AGV maximum acceleration, maximum deceleration, wheel maximum rotation angle, maximum centripetal and maximum velocity. The route has to also be possible to drive, so we have to check during the route that AGV doesn't hit the walls or other obstacles. We want to modify route such that AGV start and end velocity, start- and end point and start and end angle doesn't change during the optimization. Because all the routes are defined with the splines, in optimization we need to modify the spline's shape. Spline's shape is defined by control points, so in the optimization algorithm, we need to modify control point locations. In the optimization algorithm, we will use the properties of the genetical algorithms.

In this work, we will first go through the theory of the splines. After that, we will handle the genetical algorithm functionality at the general level. The work will also provide a brief overview of other possible approaches to optimization. This research purpose is to go through spline properties from an optimization point of view and also research the genetic algorithm suitability for solving this optimization problem. The goal is to develop

an algorithm for route optimization so that it takes account as good as possible systems' restrictions. However, our goal is not to create an algorithm that takes into account all the possible real-world and system restrictions. Instead of that, our goal is to develop an algorithm for testing the genetic algorithm's suitability for this problem. The algorithm should take into account the most common restrictions of the system such as route suitability (AGV shouldn't hit the walls), AGV maximum velocity, maximum acceleration, maximum deceleration, maximum centripetal and wheel maximum angle of rotation. The algorithm will be made so that, it is easy to add more restriction checking in to algorithm, for example checking wheel maximum rotation speed.

2 PIECEWISE POLYNOMIAL CURVES

In this chapter, we take a look into the splines. We will go through the theory on how to construct them. After that we will introduce useful properties of splines such as length of spline and derivation of splines, which are both used in constructing the theory for the optimization algorithm. To understand splines fully we have to first take a look shortly into polynomials because B-splines itself are piecewise-defined polynomial functions [2].

2.1 Polynomials

Before going into polynomials itself, we are going to shortly define the function continuity and derivability. Those both properties are needed at the spline theory but we will also use these for defining few properties for the polynomials.

Definition 2.1 (**Function continuity** [7])**.** Let $f(x)$ be the function from $I \to \mathbf{R}$, where $I \subset \mathbf{R}$, and x_0 be the point, such that $x_0 \in I$. If for any $\varepsilon > 0$ there is $\delta > 0$ such that

$$\forall x \in I, \quad |x - x_0| < \delta \implies |f(x) - f(x_0)| < \varepsilon, \tag{2.1}$$

then function f is called a **continuous at the point** x_0.

Let's next define few continuity properties for the functions.

Lemma 2.1. *Let $f(x) = x$ be the function from $\mathbf{R} \to \mathbf{R}$, then the function $f(x)$ is continuous.*

Proof. Let $f(x) = x$, such that $f : \mathbf{R} \to \mathbf{R}$. Let $\varepsilon > 0$. Let's define $\delta = \varepsilon$. For every x we have that $|x - x_0| < \delta$.

$$\begin{aligned} |f(x) - f(x_0)| &= |x - x_0| \\ &< \delta \\ &= \varepsilon. \end{aligned}$$

□

Next, we will study the continuity when summing two continuous functions together.

Lemma 2.2. *Let $f(x)$ and $g(x)$ be the continuous functions, then sum of two continuous*

functions $f(x) + g(x)$ is continuous.

Proof. Let $f(x)$ and $g(x)$ be the continuous functions, $\varepsilon = \varepsilon_1 + \varepsilon_2 > 0$ and $\delta > 0$. Because $f(x)$ is continuous, then for any $\varepsilon_1 > 0$ there is $\delta_1 = \delta/2 > 0$ such that

$$\forall x \in \mathbb{R}, \quad |x - x_0| < \delta_1 \implies |f(x) - f(x_0)| < \varepsilon_1.$$

Also, because $g(x)$ is continuous, then for any $\varepsilon_2 > 0$ there is $\delta_2 = \delta/2 > 0$ such that

$$\forall x \in \mathbb{R}, \quad |x - x_0| < \delta_2 \implies |g(x) - g(x_0)| < \varepsilon_2.$$

For every x we have that $|x - x_0| < \delta$.

$$\begin{aligned} |(f(x) + g(x)) - (f(x_0) + g(x_0))| &= |f(x) + g(x) - f(x_0) - g(x_0)| \\ &= |f(x) - f(x_0) + g(x) - g(x_0)| \\ &\leq |f(x) - f(x_0)| + |g(x) - g(x_0)| \\ &< \varepsilon_1 + \varepsilon_2 \\ &= \varepsilon. \end{aligned}$$

□

Next, we are going to define the function differentiability.

Definition 2.2 (**Function differentiability** [7])**.** Let f be the function from $I \to \mathbf{R}$, where $I \subset \mathbf{R}$, and x_0 be the point, such that $x_0 \in I$. Let f be defined at the neighbourhood of x_0. If

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{2.2}$$

exists and is finite, then f is called **differentiable at $\mathbf{x_0}$** and $f'(x_0)$ is the **derivate of f at point** x_0. Function is said to be **differentiable**, if it is differentiable at every point of its' domain. Function is said to be **continuously differentiable** if it is differentiable and its' derivative is continuous.

Now, when we have defined the function continuity and the differentiability, we can go to the polynomial theory. Polynomials are functions that are constructed by summing terms of the form ax^n. In that term, a is a coefficient of the corresponding term, x is the general variable and n is a positive integer. Highest n in function's terms defines polynomials degree. [6]

Definition 2.3 (**$\mathbf{n}$th degree polynomial**)**.** Let $n \in \mathbb{N}$ and $a \in \mathbb{R}$. n-th degree polynomial is defined such that

$$f(x) = \sum_{i=0}^{n} a_i x^i, \tag{2.3}$$

where a_i is the corresponding coefficient of the term, such that $a_n \neq 0$ if $n > 0$, and n is

the degree of the polynomial.

Next, we will go through few examples of different order polynomials.

Example 2.1. *Let's have a examples of* 0, 1 *and* 2 *degree polynomials.*

1. *0 degree polynomial:* $f(x) = a$*, where* $a \in \mathbb{R}$*.*
2. *1 degree polynomial:* $f(x) = 2x + 1$ *or* $f(x) = x$*.*
3. *2 degree polynomial:* $f(x) = 4x^2 + 1$ *or* $f(x) = x^2 + x$ *or* $f(x) = x^2 + x + 1$*.*

Polynomials have many useful properties that are useful when studying the splines theory. Next, we will introduce few of those properties and after that, we will also prove them.

Lemma 2.3 ([6])**.** *Let* $f(x)$ *be* n*th degree polynomial,* $g(x)$ *be* m*th degree polynomial and* $n, m \in \mathbb{N}$ *such that* $n < m$*. Then the following properties hold for both* f *and* g*:*

1. $f(x) + g(x)$ *is* m*th degree polynomial.*
2. $cf(x)$ *is* n*th degree polynomial, if* $c \neq 0$*.*
3. $f(x)g(x)$ *is* $m + n$*th degree polynomial.*
4. $f(x)$ *is continuous.*
5. $f(x) + g(x)$ *is continuous.*
6. $f(x)g(x)$ *is continuous.*
7. $f(x)$ *is infinitely times derivable.*

Proof. Let $f(x)$ be nth order polynomial, $g(x)$ be mth order polynomial and $n, m \in \mathbb{N}$ such that $n < m$. Let $f(x) = \sum_{i=0}^{n} a_i x^i$ and $g(x) = \sum_{i=0}^{m} b_i x^i$.

1) With straight calculation

$$\begin{aligned}
f(x) + g(x) &= \sum_{i=0}^{n} a_i x^i + \sum_{j=0}^{m} b_j x^j \\
&= \sum_{i=0}^{n} a_i x^i + \sum_{j=0}^{n} (b_j x^j) + \sum_{j=n+1}^{m} b_j x^j \\
&= \sum_{i=0}^{n} (a_i x^i + b_i x^i) + \sum_{j=n+1}^{m} b_j x^j \\
&= \sum_{i=0}^{n} ((a_i + b_i) x^i) + \sum_{j=n+1}^{m} b_j x^j \\
&= (a_0 + b_0) + (a_1 + b_1)x + \dots \\
&\quad + (a_n + b_n)x^n + b_{n+1}x^{n+1} + b_{n+2}x^{n+2} + \cdots + b_m x^m.
\end{aligned}$$

Because $f(x) + g(x)$ can be written in polynomial form and m is the highest power of x, then $f(x) + g(x)$ polynomial order is m.

2) Let $c \in \mathbb{R}$ and then by straight calculation

$$\begin{aligned} cf(x) &= c\sum_{i=0}^{n} a_i x^i \\ &= \sum_{i=0}^{n} \underbrace{ca_i}_{\text{constant}} \cdot x^i. \end{aligned}$$

Because we get form where terms are form $a_i x^i$, then $cf(x)$ is also polynom and it's order is n, if $c \neq 0$.
3) With straight calculations

$$\begin{aligned} f(x)g(x) &= \sum_{i=0}^{n} a_i x^i \sum_{j=0}^{m} b_j x^j \\ &= (a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n)(b_0 + b_1 x + b_2 x^2 + \cdots + b_m x^m) \\ &= (b_0 a_0 + b_0 a_1 x + b_0 a_2 x^2 + \cdots + b_0 a_n x^n) \\ &\quad + (b_1 x a_0 + b_1 x a_1 x + b_1 x a_2 x^2 + \cdots + b_1 x a_n x^n) + \ldots \\ &\quad + (b_m x^m a_0 + b_m x^m a_1 x + b_m x^m a_2 x^2 + \cdots + b_m x^m a_n x^n) \\ &= b_0 a_0 + b_0 a_1 x + b_0 a_2 x^2 + \cdots + b_0 a_n x^n + b_1 a_0 x + b_1 a_1 x^2 + b_1 a_2 x^3 \\ &\quad + \cdots + b_1 a_n x^{n+1} + \cdots + b_m a_0 x^m + b_m a_1 x^{m+1} + b_m a_2 x^{m+2} \\ &\quad + \cdots + b_m a_n x^{m+n} \\ &= b_0(a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n) \\ &\quad + b_1(a_0 x + a_1 x^2 + a_2 x^3 + \cdots + a_n x^{n+1}) + \ldots \\ &\quad + b_m(a_0 x^m + a_1 x^{m+1} + a_2 x^{m+2} + \cdots + a_n x^{m+n}) \\ &= b_0 \sum_{i=0}^{n} a_i x^i + b_1 \sum_{i=1}^{n+1} a_{i-1} x^i + \cdots + b_m \sum_{i=m}^{n+m} a_{i-m} x^i. \end{aligned}$$

Now by (1) and (2) $f(x)g(x)$ is polynom. The highest exponent of $f(x)g(x)$ is $m + n$, so $f(x)g(x)$ polynomial order is $m + n$.

4) We know that x is continuous everywhere. If x is continuous everywhere, then $x \cdot x$ is continuous everywhere. Now by multiplying continuous x n-times we get that x^n is continuous everywhere, for every $n \in \mathbb{N}$. We know that the sum of continuous functions is continuous then $x + x^2 + \cdots + x^n$ is continuous. Also, we know that multiplying x by constant $a \in \mathbb{R}$ is still continuous. By these facts $a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ is continuous.

5) We know that the sum of continuous functions are continuous and by 4) polynomial are continuous, then also their sum is continuous.

6) We know that multiplying two continuous gives continuous function and by 4) polynomial are continuous, then also their product is continuous.

7) Because $f(x)$ is polynom, then $f(x) = a_0 + a_1x + a_2x_2^2 + \cdots + a_nx^n$. Now with straight calculation derivate of $f(x)$ is $f'(x) = 0 + a_1 + 2a_2x_2 + \cdots + na_nx^{n-1}$. Derivate is also polynom and its degree is one degree lower than original degree. So derivating the polynom only reduces degree of the polynom by 1. If we derivate $f(x)$ n-times then $f^n(x) = (n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 2 \cdot 1)$. If we now derivate this one more time then derivate will be 0. As we know derivate of 0 is always zero, so after that all derivates are 0. □

Polynomials are simple functions that have nice properties. Because they are infinitely times differentiable they are called smooth functions which make them even more useful.

Next, we will go through the piecewise polynomials. Piecewise polynomials are important when we are dealing with B-splines because they are constructed by piecewise polynomials. Piecewise polynomials are polynomials which are defined differently in different sections. Every section is defined by the polynomial. Piecewise polynomials are not necessarily continuous or can be continuous but in that case, they are not necessarily differentiable. Next example illustrates these cases.

Example 2.2. *Let piecewise polynomial $f(x)$ be defined such that*

$$f(x) = \begin{cases} x & 0 \leq x \leq 1 \\ x+1 & x > 1. \end{cases}$$

This function is piecewise polynom, because all its parts are polynomials. This is not continuous function, because when we approach 1 from the left we get

$$\lim_{x_- \to 1} f(x) = \lim_{x_- \to 1} (x) \to 1$$

and respectively when we approach 1 from the right we get

$$\lim_{x_+ \to 1} f(x) = \lim_{x_+ \to 1} (x+1) \to 2.$$

Now let piecewice polynomial $f(x)$ be defined such that

$$f(x) = \begin{cases} x & 0 \leq x \leq 1 \\ -x+2 & x > 1. \end{cases}$$

This function is piecewise polynomial, because all its parts are polynomials. This is also

continuous function, because when we approach 1 from left we get

$$\lim_{x_- \to 1} f(x) = \lim_{x_- \to 1} (x) \to 1$$

and respectively when we approach 1 from right side we get

$$\lim_{x_+ \to 1} f(x) = \lim_{x_+ \to 1} (-x + 2) \to 1.$$

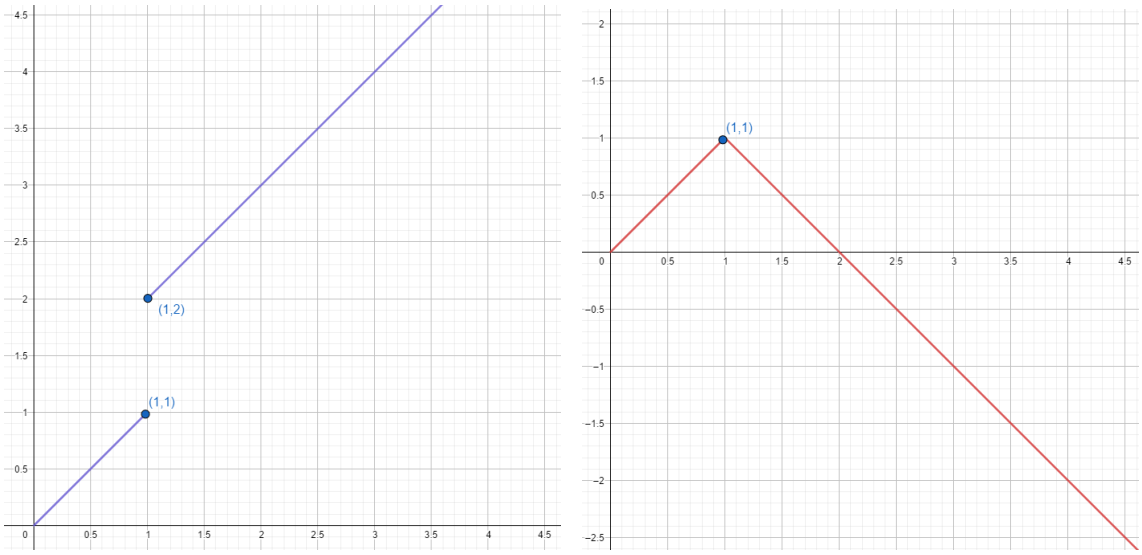
This function is not differentiable, because if $h > 0$ *we have that*

$$\begin{aligned} f'(1) &= \lim_{h \to 0} \frac{f(1+h) - f(1)}{h} \\ &= \lim_{h \to 0} \frac{-(1+h) + 2 - 1}{h} \\ &= \lim_{h \to 0} \frac{-1 - h + 1}{h} \\ &= \lim_{h \to 0} \frac{-h}{h} \\ &= \lim_{h \to 0} -1 \\ &= -1 \end{aligned}$$

and if $h < 0$

$$\begin{aligned} f'(1) &= \lim_{h \to 0} \frac{f(1+h) - f(1)}{h} \\ &= \lim_{h \to 0} \frac{(1+h) - 1}{h} \\ &= \lim_{h \to 0} \frac{-1 + h + 1}{h} \\ &= \lim_{h \to 0} \frac{h}{h} \\ &= \lim_{h \to 0} 1 \\ &= 1. \end{aligned}$$

Now because we approach different values, when approaching $x = 1$ *from left and right, then function is not differentiable at point* $x = 1$*. Next photo illustrates both functions.*



(a) *Example of not continuous piecewise polynom.*

(b) *Example of continuous piecewice polynom, which is not differentiable.*

Figure 2.1. *Illustration of two different piecewice polynom, which are both not differentiable, but second function is continuous.*

As we saw in previous example piecewice polynomials can be discontinuous. In splines one of the most important properties is that splines are defined such that piecewice polynomials are continuous.

2.2 Divided difference

For the B-splines we need to also introduce divided differences and few of its' properties.

Definition 2.4 (**Divided difference** [15])**.** Let g be a function. Then the **divided difference** is

$$[x_0, ..., x_n]g = \sum_{j=0}^{n} \frac{g(x_j)}{\prod_{k\in\{0,...,n\}\setminus j}(x_j - x_k)}. \tag{2.4}$$

Next we will introduce few properties for the divided difference, which are needed when proving the Cox De Boor recursion algorithm for the B-splines. We won't go through the theory of the divided difference, because in this research it is only needed when constructing the Cox de Boor recursion formula.

Lemma 2.4 ([3])**.** *Let g be a function. Then*

1. $[x_i]g = g(x_i)$.
2. $[x_i, ..., x_{i+k}]g = \frac{[x_{i+1},...,x_{i+k}]g - [x_i,...,x_{i+k-1}]g}{x_{i+k} - x_i}$.

Proof. (1) With straight calculation, all we need to remember is that empty product is

always one. Now by calculation

$$\begin{aligned}
[x_i]g &= \sum_{j=i}^{i} \frac{g(x_j)}{\prod_{k\in\{i\}\setminus j}(x_j - x_k)} \\
&= \frac{g(x_i)}{\prod_{k\in\{i\}\setminus i}(x_i - x_k)} \\
&= \frac{g(x_i)}{1} \\
&= g(x_i).
\end{aligned}$$

(2) Now by the previous property $[x_i]g = g(x_i)$. Let's next calculate $[x_i, x_{i+1}]g$. By straight calculation we have

$$\begin{aligned}
[x_i, x_{i+1}]g &= \sum_{j=i}^{i+1} \frac{g(x_j)}{\prod_{k\in\{i,i+1\}\setminus j}(x_j - x_k)} \\
&= \frac{g(x_i)}{\prod_{k\in\{i,i+1\}\setminus i}(x_i - x_k)} + \frac{g(x_{i+1})}{\prod_{k\in\{i,i+1\}\setminus i+1}(x_{i+1} - x_k)} \\
&= \frac{g(x_i)}{(x_i - x_{i+1})} + \frac{g(x_{i+1})}{(x_{i+1} - x_i)} \\
&= \frac{g(x_{i+1})}{(x_{i+1} - x_i)} + \frac{-g(x_i)}{(x_{i+1} - x_i)} \\
&= \frac{g(x_{i+1}) - g(x_i)}{(x_{i+1} - x_i)} \\
&= \frac{[x_{i+1}]g - [x_i]g}{x_{i+1} - x_i}.
\end{aligned}$$

Respectively $[x_i, x_{i+1}, x_{i+2}]g$ is

$$
\begin{aligned}
[x_i, x_{i+1}, x_{i+2}]g &= \sum_{j=i}^{i+2} \frac{g(x_j)}{\prod_{k\in\{i,i+1,i+2\}\setminus j}(x_j - x_k)} \\
&= \frac{g(x_i)}{\prod_{k\in\{i,i+1,i+2\}\setminus i}(x_i - x_k)} + \frac{g(x_{i+1})}{\prod_{k\in\{i,i+1,i+2\}\setminus i+1}(x_{i+1} - x_k)} \\
&\quad + \frac{g(x_{i+2})}{\prod_{k\in\{i,i+1,i+2\}\setminus i+2}(x_{i+2} - x_k)} \\
&= \frac{g(x_i)}{(x_i - x_{i+1})(x_i - x_{i+2})} + \frac{g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+1} - x_{i+2})} \\
&\quad + \frac{g(x_{i+2})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&= \frac{g(x_i)}{(x_{i+1} - x_i)(x_{i+2} - x_i)} + \frac{-g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} \\
&\quad + \frac{g(x_{i+2})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&= \frac{g(x_i)(x_{i+2} - x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&\quad + \frac{-(x_{i+2} - x_i)g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&\quad + \frac{(x_{i+1} - x_i)g(x_{i+2})}{(x_{i+1} - x_i)(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&= \frac{g(x_i)(x_{i+2} - x_{i+1}) - (x_{i+2} - x_i)g(x_{i+1}) + (x_{i+1} - x_i)g(x_{i+2})}{(x_{i+1} - x_i)(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&= \frac{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})(\frac{g(x_i)}{(x_{i+1} - x_i)} - \frac{(x_{i+2} - x_i)g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} + \frac{g(x_{i+2})}{(x_{i+2} - x_{i+1})})}{(x_{i+1} - x_i)(x_{i+2} - x_i)(x_{i+2} - x_{i+1})} \\
&= \frac{\frac{g(x_i)}{(x_{i+1} - x_i)} - \frac{(x_{i+2} - x_i)g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} + \frac{g(x_{i+2})}{(x_{i+2} - x_{i+1})}}{(x_{i+2} - x_i)} \\
&= \frac{\frac{g(x_i)(x_{i+2} - x_i)}{(x_{i+1} - x_i)(x_{i+2} - x_i)} - \frac{(x_{i+2} - x_i)g(x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} + \frac{(x_{i+1} - x_i)g(x_{i+2})}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})}}{(x_{i+2} - x_i)} \\
&= \frac{\frac{g(x_i)(x_{i+2} - x_i) - (x_{i+2} - x_i)g(x_{i+1}) + (x_{i+1} - x_i)g(x_{i+2}) - g(x_{i+1})x_{i+1} + g(x_{i+1})x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_i)}}{(x_{i+2} - x_i)} \\
&= \frac{\frac{(g(x_{i+2}) - g(x_{i+1}))(x_{i+1} - x_i) - (g(x_{i+1}) - g(x_i))(x_{i+2} - x_{i+1})}{(x_{i+1} - x_i)(x_{i+2} - x_i)}}{(x_{i+2} - x_i)} \\
&= \frac{\frac{g(x_{i+2}) - g(x_{i+1})}{x_{i+2} - x_{i+1}} - \frac{g(x_{i+1}) - g(x_i)}{x_{i+1} - x_i)}}{(x_{i+2} - x_i)} \\
&= \frac{[x_{i+1}, x_{i+2}]g - [x_i, x_{i+1}]g}{(x_{i+2} - x_i)}.
\end{aligned}
$$

Now if we continue this recursively using previous results we obtain that

$$
[x_i, \ldots, x_{i+k}]g = \frac{[x_{i+1}, \ldots, x_{i+k}]g - [x_i, \ldots, x_{i+k-1}]g}{x_{i+k} - x_i}. \qquad \square
$$

Next we will go through the Leibniz' formula for the divided difference. It is also used in construction of the recursion algorithm.

Lemma 2.5 (**Leibniz' formula** [3]). *Let g and h be functions of x. If $(gh)(x) = g(x)h(x)$ for all x, then*

$$[x_0, \ldots, x_k](gh) = \sum_{r=0}^{k} ([x_0, \ldots, x_r]g)([x_r, \ldots, x_k]h). \tag{2.5}$$

Proof. Let's prove this by induction.
The base case: Let's go through the case $[x_0](gh)$. Let's assume that for all x hold that $(gh)(x) = g(x)h(x)$. Now by straight calculation

$$\begin{aligned}
[x_0](gh) &= (gh)(x_0) \\
&= g(x_0)h(x_0) \\
&= [x_0]g[x_0]h.
\end{aligned}$$

Because $[x_0](gh) = [x_0]g[x_0]h$, then this holds for the basic case.

The induction hypothesis: Let's assume that holds

$$[x_0, \ldots, x_k](gh) = \sum_{r=0}^{k} ([x_0, \ldots, x_r]g)([x_r, \ldots, x_k]h). \tag{2.6}$$

Let's go through the couple of first steps to get the idea.
1st step:

$$\begin{aligned}
[x_0, x_1](gh) &= \frac{[x_1](gh) - [x_0](gh)}{x_1 - x_0} \\
&= \frac{(gh)(x_1) - (gh)(x_0)}{x_1 - x_0} \\
&= \frac{g(x_1)h(x_1) - g(x_0)h(x_0)}{x_1 - x_0} \\
&= \frac{g(x_1)h(x_1) + g(x_1)h(x_0) - g(x_1)h(x_0) - g(x_0)h(x_0)}{x_1 - x_0} \\
&= \frac{g(x_1)h(x_1) - g(x_1)h(x_0)}{x_1 - x_0} + \frac{g(x_1)h(x_0) - g(x_0)h(x_0)}{x_1 - x_0} \\
&= \frac{g(x_1)(h(x_1) - h(x_0))}{x_1 - x_0} + \frac{h(x_0)(g(x_1) - g(x_0))}{x_1 - x_0} \\
&= g(x_1)\frac{[x_1]h - [x_0]h}{x_1 - x_0} + h(x_0)\frac{[x_1]g - [x_0]g)}{x_1 - x_0} \\
&= [x_1]g[x_0, x_1]h + [x_0, x_1]g[x_0]h \\
&= \sum_{r=0}^{1} ([x_0, \ldots, x_r]g)([x_r, \ldots, x_1]h).
\end{aligned}$$

2nd step:

$$
\begin{aligned}
[x_0,x_1,x_2](gh) &= \frac{[x_1,x_2](gh)-[x_0,x_1](gh)}{x_2-x_0}\\
&= \frac{[x_2]h[x_1,x_2]g+[x_1,x_2]h[x_1]g-[x_1]h[x_0,x_1]g-[x_0,x_1]h[x_0]g}{x_2-x_0}\\
&= \frac{[x_1,x_2]h[x_1]g-[x_0,x_1]h[x_0]g}{x_2-x_0}+\frac{[x_2]h[x_1,x_2]g-[x_1]h[x_0,x_1]g}{x_2-x_0}\\
&= \frac{[x_1,x_2]h[x_1]g-[x_1,x_2]h[x_0]g+[x_1,x_2]h[x_0]g-[x_0,x_1]h[x_0]g}{x_2-x_0}\\
&\quad+\frac{[x_2]h[x_1,x_2]g-[x_2]h[x_0,x_1]g+[x_2]h[x_0,x_1]g-[x_1]h[x_0,x_1]g}{x_2-x_0}\\
&= \frac{[x_0]g([x_1,x_2]h-[x_0,x_1]h)}{x_2-x_0}+\frac{x_1-x_0}{x_2-x_0}\frac{[x_1,x_2]h([x_1]g-[x_0]g)}{x_1-x_0}\\
&\quad+\frac{[x_2]h([x_1,x_2]g-[x_0,x_1]g)}{x_2-x_0}+\frac{x_2-x_1}{x_2-x_0}\frac{[x_0,x_1]g([x_2]h-[x_1]h)}{x_2-x_1}\\
&= [x_0]g[x_0,x_1,x_2]h+\frac{x_1-x_0}{x_2-x_0}[x_0,x_1]g[x_1,x_2]h\\
&\quad+[x_0,x_1,x_2]g[x_3]h+\frac{x_2-x_1}{x_2-x_0}[x_0,x_1]g[x_1,x_2]h\\
&= [x_0]g[x_0,x_1,x_2]h+[x_0,x_1]g[x_1,x_2]h+[x_0,x_1,x_2]g[x_2]h\\
&= \sum_{r=0}^{2}([x_0,\dots,x_r]g)([x_r,\dots,x_2]h).
\end{aligned}
$$

Induction step: Next we prove this for the case $k+1$, at the first we apply equation (2.4), use the induction hypothesis and then we get:

$$
\begin{aligned}
[x_0,\dots,x_{k+1}](gh) &= \frac{[x_1,\dots,x_{k+1}](gh)-[x_0,\dots,x_k](gh)}{x_{k+1}-x_0}\\
&= \frac{\sum_{r=1}^{k+1}([x_1,\dots,x_r]g)([x_r,\dots,x_{k+1}]h)-\sum_{r=0}^{k}([x_0,\dots,x_r]g)([x_r,\dots,x_k]h)}{x_{k+1}-x_0}\\
&= \frac{\sum_{r=1}^{k+1}([x_1,\dots,x_r]g)([x_r,\dots,x_{k+1}]h)-\sum_{r=0}^{k}([x_0,\dots,x_r]g)([x_r,\dots,x_k]h)}{x_{k+1}-x_0}.
\end{aligned}
$$

Next we add and substract the term $\frac{\sum_{r=0}^{k}[x_0,\dots,x_r]g[x_{r+1},\dots,x_{k+1}]h}{x_{k+1}-x_0}$ this term is concluded from

the first two steps. Now we get

$$
\begin{aligned}
[x_0,\dots,x_{k+1}](gh) &= \frac{\sum_{r=1}^{k+1}[x_1,\dots,x_r]g[x_r,\dots,x_{k+1}]h - \sum_{r=0}^{k}[x_0,\dots,x_r]g[x_{r+1},\dots,x_{k+1}]h}{x_{k+1}-x_0}\\
&\quad+\frac{\sum_{r=0}^{k}[x_0,\dots,x_r]g[x_{r+1},\dots,x_{k+1}]h - \sum_{r=0}^{k}[x_0,\dots,x_r]g[x_r,\dots,x_k]h}{x_{k+1}-x_0}\\
&= \frac{\sum_{r=1}^{k+1}[x_1,\dots,x_r]g[x_r,\dots,x_{k+1}]h - \sum_{r=1}^{k+1}[x_0,\dots,x_{r-1}]g[x_r,\dots,x_{k+1}]h}{x_{k+1}-x_0}\\
&\quad+\frac{\sum_{r=0}^{k}[x_0,\dots,x_r]g[x_{r+1},\dots,x_{k+1}]h - [x_0,\dots,x_r]g[x_r,\dots,x_k]h}{x_{k+1}-x_0}\\
&= \frac{\sum_{r=1}^{k+1}[x_1,\dots,x_r]g[x_r,\dots,x_{k+1}]h - [x_0,\dots,x_{r-1}]g[x_r,\dots,x_{k+1}]h}{x_{k+1}-x_0}\\
&\quad+\frac{\sum_{r=0}^{k}[x_0,\dots,x_r]g([x_{r+1},\dots,x_{k+1}]h - [x_r,\dots,x_k]h)}{x_{k+1}-x_0}\\
&= \frac{\sum_{r=1}^{k+1}[x_r,\dots,x_{k+1}]h([x_1,\dots,x_r]g - [x_0,\dots,x_{r-1}]g)}{x_{k+1}-x_0}\\
&\quad+\frac{\sum_{r=0}^{k}[x_0,\dots,x_r]g([x_{r+1},\dots,x_{k+1}]h - [x_r,\dots,x_k]h)}{x_{k+1}-x_0}\\
&= \sum_{r=1}^{k+1}\frac{x_r-x_0}{x_{k+1}-x_0}[x_r,\dots,x_{k+1}]h\frac{[x_1,\dots,x_r]g - [x_0,\dots,x_{r-1}]g}{x_r-x_0}\\
&\quad+\sum_{r=0}^{k}\frac{x_{k+1}-x_r}{x_{k+1}-x_0}[x_0,\dots,x_r]g\frac{[x_{r+1},\dots,x_{k+1}]h - [x_r,\dots,x_k]h}{x_{k+1}-x_r}\\
&= \sum_{r=1}^{k+1}\frac{x_r-x_0}{x_{k+1}-x_0}[x_0,\dots,x_r]g[x_r,\dots,x_{k+1}]h\\
&\quad+\sum_{r=0}^{k}\frac{x_{k+1}-x_r}{x_{k+1}-x_0}[x_0,\dots,x_r]g[x_r,\dots,x_{k+1}]h.
\end{aligned}
$$

Next we open the sum equations and see if we are able to simplify expression.

$$
\begin{aligned}
[x_0,\dots,x_{k+1}](gh) &= \frac{x_1-x_0}{x_{k+1}-x_0}[x_0,x_1]g[x_1,\dots,x_{k+1}]h + \frac{x_2-x_0}{x_{k+1}-x_0}[x_0,x_1,x_2]g[x_2,\dots,x_{k+1}]h \\
&\quad + \cdots + \frac{x_{k+1}-x_0}{x_{k+1}-x_0}[x_0,\dots,x_{k+1}]g[x_{k+1}]h + \frac{x_{k+1}-x_0}{x_{k+1}-x_0}[x_0]g[x_0,\dots,x_{k+1}]h \\
&\quad + \frac{x_{k+1}-x_1}{x_{k+1}-x_0}[x_0,x_1]g[x_1,\dots,x_{k+1}]h + \cdots + \frac{x_{k+1}-x_k}{x_{k+1}-x_0}[x_0,\dots,x_k]g[x_k,x_{k+1}]h \\
&= [x_0]g[x_0,\dots,x_{k+1}]h + \frac{x_{k+1}-x_1+x_1-x_0}{x_{k+1}-x_0}[x_0,x_1]g[x_1,\dots,x_{k+1}]h \\
&\quad + \frac{x_{k+1}-x_2+x_2-x_0}{x_{k+1}-x_0}[x_0,x_1,x_2]g[x_2,\dots,x_{k+1}]h \\
&\quad + \cdots + \frac{x_{k+1}-x_k+x_k-x_0}{x_{k+1}-x_0}[x_0,\dots,x_k]g[x_k,x_{k+1}]h + [x_0,\dots,x_{k+1}]g[x_{k+1}]h \\
&= [x_0]g[x_0,\dots,x_{k+1}]h + [x_0,x_1]g[x_1,\dots,x_{k+1}]h + [x_0,x_1,x_2]g[x_2,\dots,x_{k+1}]h \\
&\quad + \cdots + [x_0,\dots,x_k]g[x_k,x_{k+1}]h + [x_0,\dots,x_{k+1}]g[x_{k+1}]h \\
&= \sum_{r=0}^{k+1}([x_0,\dots,x_r]g)([x_r,\dots,x_{k+1}]h). \qquad \square
\end{aligned}
$$

As previously said we won't go through much more theory about divided differences, because they are only needed for constructing the Cox de Boor recursion algorithm and they are not used anywhere else in this work. [13]

2.3 Splines

In this section, we are going to introduce and define the splines at the general level. Before defining the splines we need to define a few other properties, which are needed in spline definition. Let's first define the knot and the knot vector.

Definition 2.5 (**Knot** [19][2])**.** Let $f(x)$ be the piecewise defined function. Let function be defined in whole real-line such that first part is defined in section $(-\infty, t_0]$ and then next sections are defined in the following sections $[t_i, t_{i+1}], i = 0,\dots,m-1$ and let the last part be defined in the section $[t_m, \infty)$. Now the point t_i connects two parts together and that point is called as a **knot**.

Now when we have defined the knot, we will next define the knot vector. They both are a key thing at the spline theory and especially at the spline constructing.

Definition 2.6 (**Knot vector** [19])**.** Let $f(x)$ be the piecewise defined function and t_i be a ith knot of the function. The **knot vector** is the collection that contains all the knots t_i in non decreasing order, such that $t_0 \le t_1 \le t_2 \le \cdots \le t_m$. If $t_{i+1} - t_i = h \forall i$, then constructed spline is called a **uniform spline**.

Now when we have defined all the necessary things for the splines, we are able to define the spline function.

Definition 2.7 (**Spline of degree** n [2])**.** Let $S(t)$ be a function such that $S : \mathbb{R} \to \mathbb{R}$. Let also n be the degree of the spline function with the knot sequence T, where knot sequence $t_0, \ldots, t_m$ is non decreasing sequence. Function $S(t)$ is called a nth order spline function if it is defined on the entire real line and it fulfills the following three properties:

1. Function $S(t)$ is defined by maximally degree n polynomial in each knot interval $(-\infty, t_0]$,$[t_i, t_{i+1}], i = 0, 1, \ldots, m-1$ and $[t_m, \infty)$.
2. Function $S(t)$ is continuous everywhere.
3. Function's $S(t)$ maximally $n-1$ order derivatives are continuous everywhere.

In the definition, we also talk about the degree of the spline. We haven't defined the degree of the spline yet, because first, we had to define the spline. Next, we are going to define the degree of the spline.

Definition 2.8 (**Degree of spline** [2])**.** Let $S(t)$ be the spline. Then the **degree of the spline** is the highest polynomial degree, which appears in any knot interval $(-\infty, t_0]$,$[t_i, t_{i+1}], i = 0, 1, \ldots, m-1$ and $[t_m, \infty)$.

In many literature and applications, when talking about splines there can be talked about the order of the spline. Order of the spline is a different thing than the degree of the spline. The next we are going to define the order of the spline.

Definition 2.9 (**Order of spline** [2])**.** Let $S(t)$ be the spline. Let n be the degree of the spline. **Order of the spline** is the highest number of defining coefficients in defining polynomials in any knot interval $(-\infty, t_0]$,$[t_i, t_{i+1}], i = 0, 1, \ldots, m-1$ and $[t_m, \infty)$. Because the degree of the spline is n and the n degree polynomials always have $n+1$ coefficients, then order of the spline is $n+1$.

Next, we define a property that makes it possible for the user easily to change the shape of the spline without changing the order of the spline.

Definition 2.10 (**Control point** [26])**.** Point that defines the shape of the spline is called a **control point**.

Changing the control points location, always changes the shape of the spline. Splines are much used in computer graphics and computer science because they are quite easy

to construct and can be used to easily approximate complex shapes. In this work, we are going to use **basis splines (B-splines)**, which are also splines. Splines are such curves that in all cases curve doesn't go through all control points. If the curve goes through the all control points curve is said to be the **interpolating curve** and in another case, the curve is **approximating curve**. [2] In our work we don't mind if the curve goes through all the points or not, but that is still useful property especially in data science where splines can be used to data analysis.

2.4 Basis splines

In this section, we are going to go through the theory of the B-splines. B-splines are splines, but they are constructed in specific way. They are constructed by using basis functions and control points. These together form a parametrized piecewise polynomial function.

Definition 2.11 (**Basis spline** [20])**.** Let $S(t)$ be the spline. If the spline is defined in the following way:

$$S(t) = \sum_{i=0}^{n-1} \mathbf{p_i} N_{i,k}(t), \tag{2.7}$$

where $N_{i,k}(t)$ is the basis function of order k for the knot span i and $\mathbf{p_i}$ is knot span's the corresponding control point, then the spline is called a basis spline (**B-spline**).

Basis functions are constructed by using the Cox de Boor recursion algorithm. To be able to use that recursion algorithm we need to construct a knot sequence, define the control points and define the degree of the basis spline, which we are going to construct. Next, we will take a closer look into how knot vectors are constructed.

2.4.1 Constructing the knot sequence

In this section, we will go through the basics of constructing the knot vector and theory how knot vector effects on spline. There is two special knot sequences which should be mentioned. One of them is the knot vector of the form $[\dots, 0, 0, 0, 1, 1, 1, \dots]$. In this case, spline only has two knots and the spline is constructed from Bernstein polynomial. The other special knot vector is uniform knot sequence which will be used in this work and we will discuss more of that. If a knot sequence is uniform, then all knots are equidistant in vector. In this case splines are called **uniform splines**. [20]

If $t_0 < t_1 < t_2 < \cdots < t_m$, then B-spline won't necessary go through the first and last control point in all cases. In this problem we need to be sure that spline goes through the first and the last control point in all cases. This can be done by modifying knot vector, so that $t_1, \dots, t_j = t_0$, $t_i, \dots, t_{m-1} = t_m$ and every other points so that they are equidistant.

If we have this kind of knot vector, then B-spline will go through the first and last control point. In this case knot vector is called **non periodic equidistant knot vector**. [20]

Lemma 2.6. *If knot vector is constructed in the following way*

$$\begin{cases} t_i = t_0 & \text{, if } i \leq k \\ t_{i+1} - t_i = constant & k \leq i < n+2 \\ t_i = t_{k+(n+1)}, i \geq n+2, & \end{cases} \tag{2.8}$$

*then B-spline will go through the first and last control point. In this case knot vector is called an **open uniform knot vector**.*

To be able to prove the previous lemma, we need to first define the basis functions. Number of required knots depends on degree of the spline and number of control points. If spline degree is changed or control point amount is changed, knot sequence has to be constructed again. There exist relation between required number of knots, spline degree and number of control points, this relation is introduced in the following definition.

Definition 2.12. Let m be number of knots, n number of control points and k degree of the spline. Then required number of knots can be calculated as follows

$$m = n + k + 1. \tag{2.9}$$

Let's take example of creating knot vector.

Example 2.3. *Let degree of the spline be 2 and number of control points be 5, then* $n = 5, k = 2$. *In this case, required number of knots is* $m = n + k + 1 = 5 + 2 + 1 = 8$. *So now we can construct one possible knot vector* $T = [0, 0, 0, 1, 2, 3, 3, 3]$. *We will continue working with this example later when we are constructing B-splines.*

If knot sequence is not uniform and it is used to construct the B-spline, then constructed spline is called **non-uniform rational basis spline (NURBS)**. These splines are widely used in computer graphics. One strength of NURBS is the quite easy formulas for degree elevation or reduction without changing the shape of the spline and also changing the number of control points without changing the shape of the spline. [20]

2.4.2 Basis functions and constructing B-spline

This section contains theory of the B-spline construction. For the construction we will use Cox De Boor recursion formula which is based on B-spline definition given by Curry and

Schoenberg [3].

Definition 2.13 (Basis function definition [3])**.** Let t be knot sequence. The ith basis function of kth order B-spline is denoted by $N_{i,k}$ and it is defined by

$$N_{i,k}(t) := (t_{i+k} - t_i)[t_i, \ldots, t_{i+k}] \max((t-x), 0)^{k-1}, \forall x \in \mathbb{R}, \tag{2.10}$$

where the $\max((t-x), 0)$ is the truncated power function. For truncated power function we define that $(x)_+^0 = 0$ for $x < 0$.

Let's mark for the future $m_k(t) := \max((t-x), 0)^{k-1}$. For equation (2.10) we can apply lemma (2.4) and then we have

$$\begin{aligned} N_{i,k}(t) &= (t_{i+k} - t_i)[t_i, \ldots, t_{i+k}]m_k \\ &= (t_{i+k} - t_i)(\frac{[t_{i+i}, \ldots, t_{i+k}]m_k - [t_i, \ldots, t_{i+k-1}]m_k}{t_{i+k} - t_i}) \\ &= [t_{i+1}, \ldots, t_{i+k}]m_k - [t_i, \ldots, t_{i+k-1}]m_k \end{aligned}$$

By setting $k = 1$, we get basis functions for the first order B-spline and they are form

$$\begin{aligned} N_{i,1}(t) &= [t_{i+1}]m_1 - [t_i]m_1 \\ &= ([t_{i+1}] \max((t-x), 0))^{1-1} - ([t_i] \max((t-x), 0))^{1-1} \\ &= ([t_{i+1}] \max((t-x), 0))^0 - ([t_i] \max((t-x)^0, 0))^0 \\ &= \max((t-t_{i+1}), 0)^0 - \max((t-t_i), 0)^0 \end{aligned}$$

This is characteristic function by the definition of the truncated power function. This can be represented now as a partial function as follows

$$N_{i,1}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & otherwise \end{cases} \tag{2.11}$$

Now we can construct recursion formula known as Cox De Boor recursion algorithm, by using the B-spline definition. [3]

Lemma 2.7 (**Cox De Boor recursion algorithm** [3])**.** *Let $T = \{t_1, t_2, t_3, \ldots, t_m\}$ be sequence of non decreasing real numbers, then kth degree B-spline basis functions can be written*

$$N_{i,k}(t) = \omega_{i,k} N_{i,k-1}(t) + (1 - \omega_{i+1,k}) N_{i+1,k-1}(t) \tag{2.12}$$

where

$$\omega_{i,k} = \begin{cases} \frac{t-t_i}{t_{i+k-1}-t_i} & t_{i+k-1} - t_i \neq 0 \\ 0 & otherwise \end{cases} \tag{2.13}$$

and

$$N_{i,1}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & otherwise. \end{cases} \tag{2.14}$$

Proof. Proof adapts the proof of B-spline property (i) at page 90 at the [3]. First we will use lemma (2.5). This is used for the kth divided difference of a product to the particular product

$$\max(0,(t-x))^{k-1} = (t-x)\max(0,(t-x))^{k-2}.$$

Now we have

$$[t_i,\ldots,t_{i+k}]m_k = (t_i - t)[t_i,\ldots,t_{i+k}]m_{k-1} + [t_{i+1},\ldots,t_{i+k}]m_{k-1}. \tag{2.15}$$

Now by using (2.4) (a) and fact that if for all $r > i+1$ holds $[t_i,\ldots,t_r](t-x) = 0$, then $[t_i,t_{i+1}](t-x) = 1$, we have that

$$\begin{aligned}(t_i - t)[t_i,\ldots,t_{i+k}] &= (t_i - t)\frac{([t_{i+1},\ldots,t_{i+k}] - [t_i,\ldots,t_{i+k-1}])}{t_{i+k} - t_i} \\ &= \frac{t_i - t}{t_{i+k} - t_i}([t_{i+1},\ldots,t_{i+k}] - [t_i,\ldots,t_{i+k-1}]).\end{aligned}$$

Next we apply that to (2.15) and we get

$$\begin{aligned}[t_i,\ldots,t_{i+k}]m_k &= \frac{t-t_i}{t_{i+k}-t_i}[t_i,\ldots,t_{i+k-1}]m_{k-1} + \frac{t_{i+k}-t}{t_{i+k}-t_i}[t_{i+1},\ldots,t_{i+k}]m_{k-1} \\ &= \frac{t-t_i}{t_{i+k}-t_i}[t_i,\ldots,t_{i+k-1}]m_{k-1} + \frac{t_{i+k}-t}{t_{i+k}-t_i}[t_{i+1},\ldots,t_{i+k}]m_{k-1}\end{aligned}$$

Next we multiply both sides with $(t_{i+k} - t_i)$, when we get

$$
\begin{aligned}
(t_{i+k} - t_i)[t_i, \ldots, t_{i+k}]m_k &= (t_{i+k} - t_i)(\frac{t - t_i}{t_{i+k} - t_i}[t_i, \ldots, t_{i+k-1}]m_{k-1} \\
&\quad + \frac{t_{i+k} - t}{t_{i+k} - t_i}[t_{i+1}, \ldots, t_{i+k}]m_{k-1} \\
N_{i,k} &= (t - t_i)[t_i, \ldots, t_{i+k-1}]m_{k-1} + (t_{i+k} - t)[t_{i+1}, \ldots, t_{i+k}]m_{k-1} \\
&= (t - t_i)\frac{t_{i+k-1} - t_i}{t_{i+k-1} - t_i}[t_i, \ldots, t_{i+k-1}]m_{k-1} \\
&\quad + (t_{i+k} - t)\frac{t_{i+1+k-1} - t_{i+1}}{t_{i+1+k-1} - t_{i+1}}[t_{i+1}, \ldots, t_{i+k}]m_{k-1} \\
&= \frac{t - t_i}{t_{i+k-1} - t_i}(t_{i+k-1} - t_i)[t_i, \ldots, t_{i+k-1}]m_{k-1} \\
&\quad + \frac{t_{i+k} - t}{t_{i+1+k-1} - t_{i+1}}(t_{i+1+k-1} - t_{i+1})[t_{i+1}, \ldots, t_{i+k}]m_{k-1} \\
&= \frac{t - t_i}{t_{i+k-1} - t_i}B_{i,k-1} + \frac{t_{i+k} - t}{t_{i+1+k-1} - t_{i+1}}N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + \frac{t_{i+k} - t}{t_{i+1+k-1} - t_{i+1}}N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + (\frac{t_{i+k} - t_{i+1} - t + t_{i+1}}{t_{i+1+k-1} - t_{i+1}})N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + (\frac{t_{i+1+k-1} - t_{i+1} - (t - t_{i+1})}{t_{i+1+k-1} - t_{i+1}})N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + (\frac{t_{i+1+k-1} - t_{i+1}}{t_{i+1+k-1} - t_{i+1}} - \frac{t - t_{i+1}}{t_{i+1+k-1} - t_{i+1}})N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + (1 - \frac{t - t_{i+1}}{t_{i+1+k-1} - t_{i+1}})N_{i+1,k-1} \\
&= \omega_{i,k}N_{i,k-1} + (1 - \omega_{i+1,k})N_{i+1,k-1}. \qquad \square
\end{aligned}
$$

Cox de Boor recursion algorithm is used for creating the basis functions for constructing the B-splines. Basis functions calculation only requires knot vector. Next we will go through one spline construction by hand. Calculating the spline by hand requires much calculation. Operations in the process are simple, but when calculating the basis functions by hand it is really easy to get confused by indexes.

Example 2.4 (Continue example 2.3)**.** *We constructed knot vector* $T = [0, 0, 0, 1, 2, 3, 3, 3]$ *in example 2.3. Now we will calculate basis functions by using Cox De Boor recursion algorithm. Because the order of the spline was 3 (*$k = 2$*), we have to calculate all basis functions of order 1 and 2 to be able to calculate 3rd order basis functions.*

3rd order basis functions.

$$
\begin{aligned}
N_{0,3} &= \frac{t-0}{0-0}N_{0,2}(t) + \frac{1-t}{1-0}N_{1,2}(t) \\
&= (1-t)N_{1,2}(t) \\
&= (1-t)\big[\frac{t-0}{0-0}N_{0,1}(t) + \frac{1-t}{1-0}N_{2,1}(t)\big] \\
&= (1-t)(1-t)N_{2,1}(t) \\
&= (1-t)^2 N_{2,1}(t) \\
&= (1-2t+t^2)N_{2,1}(t) \\
N_{1,3} &= \frac{t-0}{1-0}N_{1,2}(t) + \frac{2-t}{2-0}N_{2,2}(t) \\
&= tN_{1,2}(t) + \frac{2-t}{2}N_{2,2}(t) \\
&= t\big[\frac{t-0}{0-0}N_{1,1}(t) + \frac{1-t}{1-0}N_{2,1}(t)\big] + \frac{2-t}{2}\big[\frac{t-0}{1-0}N_{2,1}(t) + \frac{2-t}{2-1}N_{3,1}(t)\big] \\
&= t - t^2 N_{2,1}(t) + \frac{2-t}{2}\big[tN_{2,1}(t) + (2-t)N_{3,1}(t)\big] \\
&= t - t^2 N_{2,1}(t) + \frac{2t-t^2}{2}N_{2,1}(t) + \frac{(2-t)^2}{2}N_{3,1}(t) \\
&= (t - t^2 + \frac{2t-t^2}{2})N_{2,1}(t) + \frac{(2-t)^2}{2}N_{3,1}(t) \\
&= (\frac{2t-2t^2}{2} + \frac{2t-t^2}{2})N_{2,1}(t) + \frac{4-4t+t^2}{2}N_{3,1}(t) \\
&= \frac{4t-3t^2}{2}N_{2,1}(t) + \frac{4-4t+t^2}{2}N_{3,1}(t)
\end{aligned}
$$

$$
\begin{aligned}
N_{2,3} &= \frac{t-0}{2-0}N_{2,2}(t) + \frac{3-t}{3-1}N_{3,2}(t) \\
&= \frac{t}{2}N_{2,2}(t) + \frac{3-t}{2}N_{3,2}(t) \\
&= \frac{t}{2}\big[\frac{t-0}{1-0}N_{2,1}(t) + \frac{2-t}{2-1}N_{3,1}(t)\big] + \frac{3-t}{2}\big[\frac{t-1}{2-1}N_{3,1}(t) + \frac{3-t}{3-2}N_{4,1}(t)\big] \\
&= \frac{t^2}{2}N_{2,1}(t) + (\frac{2t-t^2}{2} + \frac{3-t}{2}(t-1))N_{3,1}(t) + \frac{3-t}{2}(3-t)N_{4,1}(t) \\
&= \frac{t^2}{2}N_{2,1}(t) + (\frac{2t-t^2}{2} + \frac{3t-3-t^2+t}{2})N_{3,1}(t) + \frac{(3-t)^2}{2}N_{4,1}(t) \\
&= \frac{t^2}{2}N_{2,1}() + (\frac{2t-t^2}{2} + \frac{4t-3-t^2}{2})N_{3,1}(t) + \frac{(3-t)^2}{2}N_{4,1}(t) \\
&= \frac{t^2}{2}N_{2,1}(t) + \frac{6t-2t^2-3}{2}N_{3,1}(t) + \frac{9-6t+t^2}{2}N_{4,1}(t) \\
N_{3,3} &= \frac{t-1}{3-1}N_{3,2}(t) + \frac{3-t}{3-2}N_{4,2}(t) \\
&= \frac{t-1}{2}N_{3,2}(t) + (3-t)N_{4,2}(t) \\
&= \frac{t-1}{2}\big[\frac{t-1}{2-1}N_{3,1}(t) + \frac{3-t}{3-2}N_{4,1}(t)\big] + (3-t)\big[\frac{t-2}{3-2}N_{4,1}(t) + \frac{3-t}{3-3}N_{5,1}(t)\big] \\
&= \frac{(t-1)^2}{2}N_{3,1}(t) + (3-t)\frac{t-1}{2}N_{4,1}(t) + (3-t)(t-2)N_{4,1}(t) \\
&= \frac{(t-1)^2}{2}N_{3,1}(t) + (3-t)(\frac{t-1}{2} + (t-2))N_{4,1}(t) \\
&= \frac{t^2-2t+1}{2}N_{3,1}(t) + (3-t)(\frac{3t-5}{2})N_{4,1}(t) \\
&= \frac{t^2-2t+1}{2}N_{3,1}(t) + (\frac{14t-3t^2-15}{2})N_{4,1}(t) \\
N_{4,3} &= \frac{t-2}{3-2}N_{4,2}(t) + \frac{3-t}{3-3}N_{5,2}(t) \\
&= (t-2)N_{4,2}(t) \\
&= (t-2)\big[\frac{t-2}{3-2}N_{4,1}(t) + \frac{3-t}{3-3}N_{5,1}(t)\big] \\
&= (t-2)^2N_{4,1}(t) \\
&= (t^2-4t+4)N_{4,1}(t).
\end{aligned}
$$

So the basis functions are partial polynomial functions which are defined in different intervals.

$$N_{0,3} = \begin{cases} 1 - 2t + t^2 & 0 \leq t \leq 1 \\ 0 & otherwise \end{cases}$$

$$N_{1,3} = \begin{cases} \frac{4t-3t^2}{2} & 0 \leq t \leq 1 \\ \frac{4-4t+t^2}{2} & 1 \leq t \leq 2 \\ 0 & otherwise \end{cases}$$

$$N_{2,3} = \begin{cases} \frac{t^2}{2} & 0 \leq t \leq 1 \\ \frac{6t-2t^2-3}{2} & 1 \leq t \leq 2 \\ \frac{9-6t+t^2}{2} & 2 \leq t \leq 3 \\ 0 & otherwise \end{cases}$$

$$N_{3,3} = \begin{cases} \frac{t^2-2t+1}{2} & 1 \leq t \leq 2 \\ \frac{14t-3t^2-15}{2} & 2 \leq t \leq 3 \\ 0 & otherwise \end{cases}$$

$$N_{4,3} = \begin{cases} t^2 - 4t + 4 & 2 \leq t \leq 3 \\ 0 & otherwise. \end{cases}$$

Next figure will show how these basis functions behave as u changes.

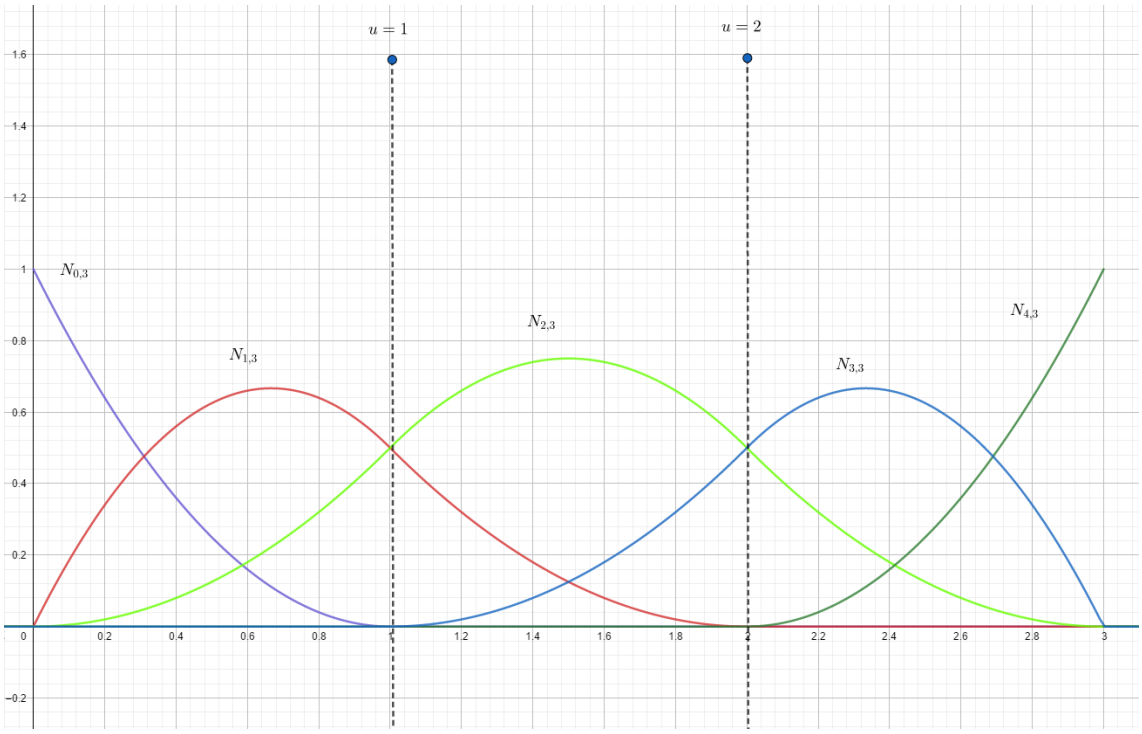


Figure 2.2. *Basis functions of the example.*

Now we construct the spline function.

$$
\begin{aligned}
S(t) &= \sum_{i=0}^{4} \mathbf{p_i} N_{i,3}(t) \\
&= \mathbf{p_0} N_{0,3}(t) + \mathbf{p_1} N_{1,3}(t) + \mathbf{p_2} N_{2,3}(t) + \mathbf{p_3} N_{2,3}(t) + \mathbf{p_4} N_{4,4}(t) \\
&= [1,1] N_{0,3}(t) + [2,2] N_{1,3}(t) + [3,2] N_{2,3}(t) + [4,1] N_{2,3}(t) + [5,0] N_{4,4}(t) \\
&= \begin{cases} [1 + 2t - \frac{1}{2}t^2, 1 + 2t - t^2] & 0 \leq t \leq 1 \\ [t + \frac{3}{2}, -\frac{t^2}{2} + t + \frac{3}{2}] & 1 \leq t \leq 2 \\ [\frac{1}{2}t^2 - t + \frac{7}{2}, -\frac{1}{2}t^2 + t + \frac{3}{2}] & 2 \leq t \leq 3. \end{cases}
\end{aligned}
$$

Next figure will illustrate shape of the spline. Control points are marked also into the figure.

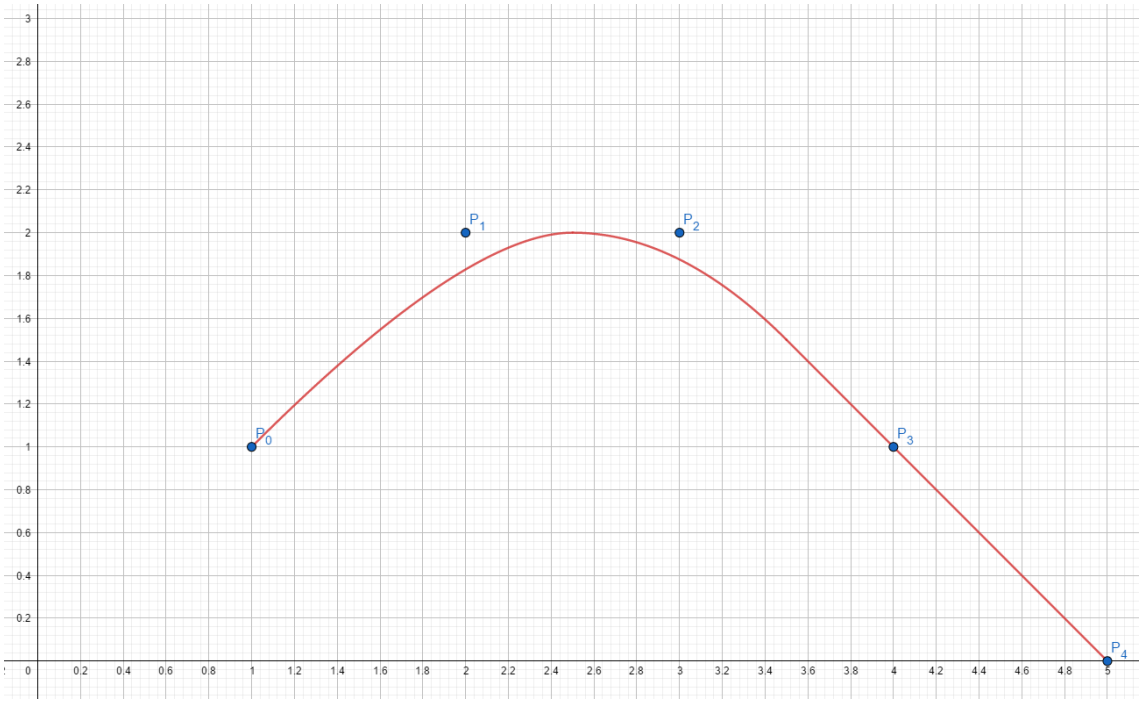


Figure 2.3. *Spline functions of the example.*

Now we have defined everything to be able to prove the lemma (2.6).

Proof of lemma (2.6). Let knot vector be $[t_0, ..., t_0, t_{k+1}, ..., t_{k+1}, t_{k+n+1}, ..., t_{k+n+1}]$ and the control point vector be $P = [p_0, ..., p_{n-1}]$. Let's calculate the first nonzero basis function

for kth order spline.

$$\begin{aligned}
N_{0,k} &= \frac{t-t_0}{t_{k-1}-t_0}N_{0,k-1}(t) + (1-\frac{t-t_1}{t_k-t_1})N_{1,k-1} \\
&= \frac{t-t_0}{t_0-t_0}N_{0,k-1}(t) + (1-\frac{t-t_1}{t_k-t_1})N_{1,k-1} \\
&= (1-\frac{t-t_1}{t_0-t_1})N_{1,k-1} \\
&= (1-\frac{t-t_0}{t_0-t_0})N_{1,k-1} \\
&= N_{1,k-1} \\
&\vdots \\
&= N_{2,k-2} \\
&\vdots \\
&= N_{k-1,k-(k-1)} \\
&= N_{k-1,1} \\
&= 1.
\end{aligned}$$

Respectively the last nonzero basis function for kth order spline is

$$\begin{aligned}
N_{n+2,k} &= \frac{t-t_{n+2}}{t_{n+2+k-1}-t_{n+2}}N_{n+2,k-1}(t) + (1-\frac{t-t_{n+3}}{t_{n+2+k}-t_{n+2}})N_{n+3,k-1} \\
&= \frac{t-t_{n+1+k}}{t_{n+1+k}-t_{n+1+k}}N_{n+2,k-1}(t) + (1-\frac{t-t_{n+3}}{t_{n+2+k}-t_{n+2}})N_{n+3,k-1} \\
&= (1-\frac{t-t_{k+(n+1)}}{t_{k+(n+1)}-t_{k+(n+1)}})N_{n+3,k-1} \\
&= 1 \cdot N_{n+3,k-1} \\
&\vdots \\
&= N_{n+4,k-2} \\
&\vdots \\
&= N_{(k-1)+n+2,k-(k-1)} \\
&= N_{k+n+1,1} \\
&= 1.
\end{aligned}$$

Now, when $t = t_0$, then $S(0) = p_0$ and respectively when $t = t_{k+n+1}$, then $S(t_{k+n+1}) = p_{n-1}$. □

Changing the control points location always changes the shape of the spline. Changing one control point location affects only the previous part, the current part and the next part of the spline. This means that if we have many control points we only affect small parts of the spline and it makes possible to make many small curves during the spline. Instead, if

we have a small number of control points we will affect the longer part of the spline and we are not able to create that much curves during the spline. In this system, we will need the way to lock the input and output angle of the route. Input and output angle are locked by not changing the first two and last two control points. In this way, any change in control points doesn't affect the first or last control point area and input and output angles remain the same. Locking the first two and last two control points means that spline has to have at least 5 control point to get algorithm work. [22]

2.5 Length of spline

To be able to determine drive time of the spline we need to be able to calculate length of the spline. There is few different ways to determine the length of the spline. We will introduce two of them. The first method is easier to implement into software algorithm and that's why it is introduced and the other one is more theoretical approach to the length of spline. That approach is not easy to implement to the algorithm, because this approach needs more accurate initial conditions and it has more complicated calculations.

2.5.1 Multiple linear segments length approximation

One way to calculate spline length is to divide spline into arbitrary small parts and calculate distance of the parts. After that we get spline length by summing all parts distances together. This method is illustrated in the following figure. Figure illustrates how linear segment approximation works, when number of segments is increased.

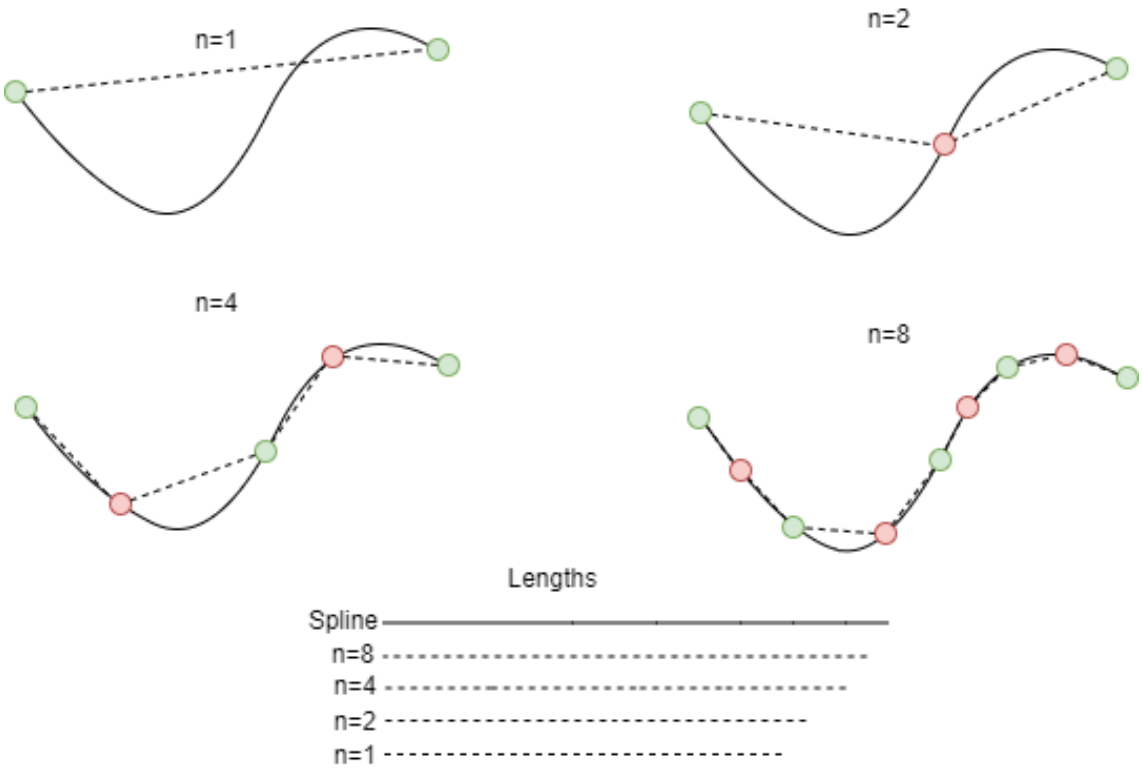


Figure 2.4. *Example of dividing spline into parts and calculating straight distance between points.*

We can create general theorem for approximating spline length by linear segment approximation method.

Theorem 2.8. *Let $S(t)$ be a arbitrary spline function and let $S(t)$ be defined in $[t_0, t_n]$, where t_0 is starting point of spline and t_n is end point of spline. Let metric to be usual metric, so that $d(\mathbf{v_1}, \mathbf{v_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, where $v_1 = x_1\mathbf{i} + y_1\mathbf{j}$ and $v_2 = x_2\mathbf{i} + y_2\mathbf{j}$. Length of the spline can be approximated as follows*

$$l(S(t)) \geq \sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))). \tag{2.16}$$

Proof. Let $S(t)$ be a arbitrary spline function and let $S(t)$ be defined in $[t_0, t_n]$, where t_0 is starting point of spline and t_n is end point of spline. Let metric be usual metric, such that $d(\mathbf{v_1}, \mathbf{v_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Let's divide interval $[t_0, t_n]$ in to equidistant n-parts. Now length of each part is $\frac{t_n - t_0}{n}$. Start point of the i-th part can be calculated such that we add i times length of each part in to the starting point t_0 as follows

$$t_0 + \frac{t_n - t_0}{n} \cdot i.$$

End point can be calculated in same way but we only add one more part length as follows

$$t_0 + \frac{t_n - t_0}{n} \cdot (i+1).$$

Distance between start and end point of the spline is

$$d(S(t_0), S(t_n)).$$

Now by the properties of the metric space [27] we have that

$$\begin{aligned} d(S(t_0), S(t_n)) &\leq d(S(t_0), S(t_0 + \frac{t_n - t_0}{2})) + d(S(t_0 + \frac{t_n - t_0}{2}, S(t_1))) \\ &\leq d(S(t_0), S(t_0 + \frac{t_n - t_0}{4})) + d(S(t_0 + \frac{t_n - t_0}{4}), S(t_0 + \frac{t_n - t_0}{2})) \\ &\quad + d(S(t_0 + \frac{t_n - t_0}{2}), S(t_0 + \frac{3(t_n - t_0)}{4})) + d(S(t_0 + \frac{3(t_n - t_0)}{4}), S(t_1))). \end{aligned}$$

If we continue this many times, then we have

$$\begin{aligned} d(S(t_0), S(t_n)) &\leq d(S(t_0), S(t_0 + \frac{t_n - t_0}{n})) + d(S(t_0 + \frac{t_n - t_0}{n}), S(t_0 + \frac{t_n - t_0}{n} \cdot 2)) + ... \\ &\quad + d(S(t_0 + \frac{t_n - t_0}{n} \cdot (n-2)), S(t_0 + \frac{3(t_n - t_0)}{n} \cdot (n-1))) \\ &\quad + d(S(t_0 + \frac{(t_n - u_0)}{n} \cdot (n-1)), S(t_n))) \\ &\leq d(S(t_0 + \frac{t_n - t_0}{n} \cdot 0), S(t_0 + \frac{t_n - t_0}{n})) \\ &\quad + d(S(t_0 + \frac{t_n - t_0}{n}), S(t_0 + \frac{t_n - t_0}{n} \cdot 2)) + ... \\ &\quad + d(S(t_0 + \frac{t_n - t_0}{n} \cdot (n-2)), S(t_0 + \frac{3(t_n - t_0)}{n} \cdot (n-1))) \\ &\quad + d(S(t_0 + \frac{(t_n - t_0)}{n} \cdot (n-1)), S(t_0 + \frac{t_n - t_0}{n} \cdot n))) \\ &\leq \sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))). \end{aligned}$$

If $n \to \infty$, then length of each part $\frac{t_n - t_0}{n} \xrightarrow{n \to \infty} 0$. This means that if we get big enough $n < \infty$, then distance between spline points tends to spline length of that part. Because of that, we can select big enough n such that when we go through all the parts of the spline and sum the distances between points we get the distance of spline. So for the big enough n we have that

$$\sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))) = l(S(t)). \tag{2.17}$$

Now by the metric space properties [27] we have that

$$\begin{aligned}
l(S(t)) &= \sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))) \\
&= d(S(t_0 + \frac{t_n - t_0}{n} \cdot 0), S(t_0 + \frac{t_n - t_0}{n})) \\
&\quad + d(S(t_0 + \frac{t_n - t_0}{n}), S(t_0 + \frac{t_n - t_0}{n} \cdot 2)) + \dots \\
&\quad + d(S(t_0 + \frac{t_n - t_0}{n} \cdot (n-2)), S(t_0 + \frac{3(t_n - t_0)}{n} \cdot (n-1))) \\
&\quad + d(S(t_0 + \frac{(t_n - t_0)}{n} \cdot (n-1)), S(t_0 + \frac{t_n - t_0}{n} \cdot n))) \\
&\geq d(S(t_0 + \frac{t_n - t_0}{n-1} \cdot 0), S(t_0 + \frac{t_1 - t_0}{n-1})) \\
&\quad + d(S(t_0 + \frac{t_n - t_0}{n-1}), S(t_0 + \frac{t_n - t_0}{n-1} \cdot 2)) + \dots \\
&\quad + d(S(t_0 + \frac{t_n - t_0}{n-1} \cdot (n-3)), S(t_0 + \frac{3(t_n - t_0)}{n-1} \cdot (n-2))) \\
&\quad + d(S(t_0 + \frac{(t_n - t_0)}{n-1} \cdot (n-2)), S(t_0 + \frac{t_n - t_0}{n-1} \cdot (n-1))) \\
&= \sum_{i=0}^{n-2} d(S(t_0 + \frac{t_1 - t_0}{n-1} \cdot i), S(t_0 + \frac{t_1 - t_0}{n-1} \cdot (i+1))).
\end{aligned}$$

□

This method always gives the approximation of the length of the curve, only when n tends to infinity, the length of the curve tends to real distance of the curve. In many cases, n doesn't need to be big to get good enough approximation of the length. Next, we will take look into maximum difference to the real distance of the curve.

Lemma 2.9. *The maximal error in multiple linear segments length approximation is*

$$err_{max} = l(S(t)) - d(S(t_0), S(t_n)). \tag{2.18}$$

Proof. Let $S(t)$ be spline where $t \in [t_0, t_n]$. Let $l(S(t))$ be length of the spline. Now, the error is

$$err = l(S(t)) - \sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))).$$

Now if n is big enough then $\sum_{i=0}^{n-1} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))) = l(S(t))$ and then the error is 0. Because

$$d(S(t_0), S(t_n)) \leq d(S(t_0)), S(t_0 + \frac{t_n - t_0}{2} \cdot 1) + d(S(t_0 + \frac{t_n - t_0}{2} \cdot 1), S(t_0 + \frac{t_n - t_0}{2} \cdot 2,$$

then smallest distance is attained if $n = 1$. So now maximum error is

$$
\begin{aligned}
err_{max} &= l(S(t)) - \sum_{i=0}^{0} d(S(t_0 + \frac{t_n - t_0}{n} \cdot i), S(t_0 + \frac{t_n - t_0}{n} \cdot (i+1))) \\
err_{max} &= l(S(t)) - d(S(t_0 + \frac{t_n - t_0}{1} \cdot 0), S(t_0 + \frac{t_n - t_0}{1} \cdot (0+1))) \\
err_{max} &= l(S(t)) - d(S(t_0), S(t_0 + t_n - t_0)) \\
err_{max} &= l(S(t)) - d(S(t_0), S(t_n)).
\end{aligned}
$$

□

Because we can find the minimum for the error at the length of the curve, then with this method curve length can't ever differ from the real length more than the maximum error. Because maximum error is attained when $n = 1$, then by growing the n always gives the better approximation or at least the same than previous length were.

Computing the distance needs much resources when spline is divided in to really small parts. This method is good when rough approximation is enough or there is much resources available. Let's take one example, how to calculate spline distance with this method.

Example 2.5. *Let's continue from example 2.4, where we constructed B-spline. Now we can calculate length of the that spline. Constructed spline was*

$$
S(t) = \begin{cases} [1 + 2t - \frac{1}{2}t^2, 1 + 2t - t^2] & 0 \leq t \leq 1 \\ [t + \frac{3}{2}, -\frac{t^2}{2} + t + \frac{3}{2}] & 1 \leq t \leq 2 \\ [\frac{1}{2}t^2 - t + \frac{7}{2}, -\frac{1}{2}t^2 + t + \frac{3}{2}] & 2 \leq t \leq 3 \end{cases}
$$

and knot vector was $T = [0, 0, 0, 1, 2, 3, 3, 3]$*. Now length can be approximated by formula* (2.16)*. Let's calculate length by different values* $n = 1, 6, 100$*. Now, when* $n = 1$*, we have*

that

$$\begin{aligned}
l_1 &= \sum_{i=0}^{1} d(S(0+\frac{3-0}{2}\cdot i), S(0+\frac{3-0}{2}\cdot(i+1))) \\
&= d(S(0+\frac{3-0}{1}\cdot 0), S(0+\frac{3-0}{1}\cdot(0+1))) \\
&= d(S(0), S(0+3\cdot 1)) \\
&= d(S(0), S(3)) \\
&= d([1+2\cdot 0-\frac{1}{2}\cdot 0^2, 1+2\cdot 0-0^2], [\frac{1}{2}3^2-3+\frac{7}{2}, -\frac{1}{2}3^2+3+\frac{3}{2}]) \\
&= d([1,1], [\frac{9}{2}-\frac{6}{2}+\frac{7}{2}, -\frac{9}{2}+\frac{6}{2}+\frac{3}{2}]) \\
&= d([1,1], [\frac{10}{2}, 0]) \\
&= d([1,1], [5,0]) \\
&= \sqrt{(5-1)^2+(0-1)^2} \\
&= \sqrt{4^2+1} \\
&= \sqrt{16+1} \\
&= \sqrt{17} \\
&\approx 4,12.
\end{aligned}$$

When $n = 6$, we have that

$$\begin{aligned}
l_6 &= \sum_{i=0}^{5} d(S(0+\frac{3-0}{6}\cdot i), S(0+\frac{3-0}{6}\cdot(i+1))) \\
&= d(S(0+\frac{3-0}{6}\cdot 0), S(0+\frac{3-0}{6}\cdot(0+1))) \\
&\quad + d(S(0+\frac{3-0}{6}\cdot 1), S(0+\frac{3-0}{6}\cdot 2)) + ... \\
&\quad + d(S(0+\frac{3-0}{6}\cdot 5), S(0+\frac{3-0}{6}\cdot 6)) \\
&= d(S(0), S(\frac{3}{6})) + d(S(\frac{3}{6}), S(\frac{3}{3})) + ... + d(S(\frac{15}{6}), S(3)) \\
&= d(S(0), S(\frac{1}{2})) + d(S(\frac{1}{2}), S(1)) + ... + d(S(\frac{5}{2}), S(3))
\end{aligned}$$

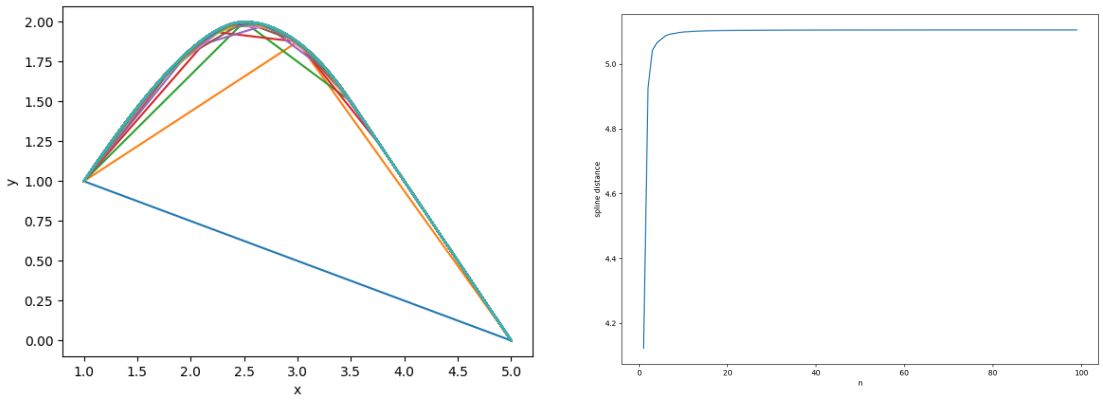
Next, we have to calculate values of spline at points $t = \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}$.

$$\begin{aligned}
S(\frac{1}{2}) &= [1 + 2\frac{1}{2} - \frac{1}{2}(\frac{1}{2})^2, 1 + 2\frac{1}{2} - (\frac{1}{2})^2] \\
&= [1 + 1 - \frac{1}{2}\frac{1}{4}, 1 + 1 - \frac{1}{4}] \\
&= [\frac{15}{8}, \frac{7}{4}] \\
S(1) &= [1 + 2 \cdot 1 - \frac{1}{2}1^2, 1 + 2 \cdot 1 - 1^2] \\
&= [1 + 2 \cdot 1 - \frac{1}{2}1^2, 1 + 2 \cdot 1 - 1^2] \\
&= [1 + 2 - \frac{1}{2}, 1 + 2 - 1] \\
&= [\frac{5}{2}, 2] \\
S(\frac{3}{2}) &= [\frac{3}{2} + \frac{3}{2}, -\frac{(\frac{3}{2})^2}{2} + \frac{3}{2} + \frac{3}{2}] \\
&= [\frac{6}{2}, -\frac{\frac{9}{4}}{2} + \frac{6}{2}] \\
&= [3, -\frac{9}{8} + \frac{24}{8}] \\
&= [3, \frac{15}{8}] \\
S(2) &= [2 + \frac{3}{2}, -\frac{2^2}{2} + 2 + \frac{3}{2}] \\
&= [\frac{7}{2}, \frac{3}{2}] \\
S(\frac{5}{2}) &= [\frac{1}{2}(\frac{5}{2})^2 - \frac{5}{2} + \frac{7}{2}, -\frac{1}{2}(\frac{5}{2})^2 + \frac{5}{2} + \frac{3}{2}] \\
&= [\frac{1}{2}\frac{25}{4} + \frac{2}{2}, -\frac{1}{2}\frac{25}{4} + \frac{8}{2}] \\
&= [\frac{25}{8} + \frac{8}{8}, -\frac{25}{8} + \frac{32}{8}] \\
&= [\frac{33}{8}, \frac{7}{8}].
\end{aligned}$$

Now, we can calculate the distance

$$\begin{aligned}
l_6 &= d([1,1], [\frac{15}{8}, \frac{7}{4}]) + d([\frac{15}{8}, \frac{7}{4}], [\frac{5}{2}, 2]) \\
&+ d([\frac{5}{2}, 2], [3, \frac{15}{8}]) + d([3, \frac{15}{8}], [\frac{7}{2}, \frac{3}{2}]) \\
&+ d([\frac{7}{2}, \frac{3}{2}], [\frac{33}{8}, \frac{7}{8}]) + d([\frac{33}{8}, \frac{7}{8}], [5, 0]) \\
&= \frac{\sqrt{85}}{8} + \frac{\sqrt{29}}{8} + \frac{\sqrt{17}}{8} + \frac{5}{8} + \frac{5}{4\sqrt{2}} + \frac{7}{4\sqrt{2}} \\
&\approx 5,087.
\end{aligned}$$

$n = 100$ *case is calculated with computer, but its logic is still the same. In* $n = 100$ *case we have that* $l_{100} = 5,105$.



(a) *Example how spline length calculation changes as n grows from 1 to 100.*

(b) *Example how spline lengths changes as n grows.*

Figure 2.5. *Illustration how changing the n effects on spline length.*

Another way to calculate length of the spline is to use so called "lenght of the parametrized curve". We will take a look into this method in the following subsection.

2.5.2 The length of the parametrized curve

Before we are going to introduce other way for calculating the length of the spline we have to make clear few definitions. The another, general definition for the curve length is introduced, because it is needed, when constructing the curvature theory. The first we define a parametrized curve.

Definition 2.14 (**Parametrized curve** [11])**.** Let $t \in [a, b]$, where $a, b \in \mathbb{R}$ and $a < b$. Let $f(t) = [x(t), y(t)]$. Now $x(t)$ and $y(t)$ are **component functions** of $f(t)$ and t is said to be the parameter. The **curve** is the image of the f, in the other words the curve C is the set of points such that $C = [x(t), y(t)] : t \in [a, b]$. In this case f is said to be a **parametric representation of the curve** C and the curve C is said to be the parametrized by the function f.

Because the spline is a function with parametric representation, then it is a curve. In this system our splines are curves in $\mathbb{R}^2$. Next, we will define the curve smoothness.

Definition 2.15 ([11])**.** Let C be the curve and t be the parametrization of the curve C. C is said to be **smooth** if there exist a continuously differentiable parametrisation c such that $c'(t) \neq (0, 0) \quad \forall t \in [a, b]$.

Next, we will define the reparametrization of the curve.

Definition 2.16 ([5])**.** Let C be the curve and $\mathbf{r}(t)$ be the parametrization of the curve C,

such that $t \in [a, b]$. Curves' C reparametrization is an invertible differentiable function $\phi : [c, d] \to [a, b]$, $t = \phi(s)$. From that we define a new parametrization

$$\widetilde{\mathbf{r}}(s) = \mathbf{r}(\phi(s)), \tag{2.19}$$

where $s \in [c, d]$, $c = \phi^{-1}(a)$ and $d = \phi^{-1}(b)$.

Finally, we will define the length of the parametrized curve.

Definition 2.17 (Length of parametrized curve [5])**.** Let C be the curve, with injective parametrization $\mathbf{c}(t)$, where $t \in [t_0, t_n]$. Now length of the curve $l(C)$ can be calulated by

$$l(C) = \int_{t_0}^{t_n} ||\mathbf{c}'(t)||dt. \tag{2.20}$$

The length of the curve doesn't depend on the parametrization, if the parametrization is smooth.

Important thing when evaluating length of the parametrized curve is that the parametrization is not allowed to go through same values in coordinate system twice when t changes. This means that curve cannot change the direction or create any sort of loops. Because of that, curves' x-parameter function derivate have to be all the time positive or the negative, it cannot change during the curve. When derivate is positive, curve is moving to right and x-values are growing, when t grows, and if derivate is negative, then curve is moving left and x-values are decreasing when t grows. [5]

Definition 2.18. Let $\mathbf{s}(t)$ be a parametrized function, where $t \in [t_0, t_n]$. If for all $t_1, t_2 \in [t_0, t_n]$

$$\mathbf{s}(t_1) = \mathbf{s}(t_2) \Rightarrow t_1 = t_2, \tag{2.21}$$

then $\mathbf{s}(t)$ is injective.

Let's go through few basic examples of injective and non-injective functions.

Example 2.6. *Let* $\mathbf{s}(t) = [t - 1, t + 2]$*. Now let's take* $t_1, t_2 \in [t_0, t_n]$*,such that* $t_1 = a$ *and* $t_2 = b$*. Now by the straight calculation*

$$\begin{aligned} \mathbf{s}(t_1) &= \mathbf{s}(t_2) \\ \mathbf{s}(a) &= \mathbf{s}(b) \\ [a - 1, a + 2] &= [b - 1, b + 2]. \end{aligned}$$

We know that this is true only when both vectors elements equal, so

$$\begin{aligned}
&[a-1, a+2] = [b-1, b+2] \\
\Rightarrow\ &a-1 = b-1 \wedge a+2 = b+2 \\
&a = b \wedge a = b \\
&\Rightarrow t_1 = t_2.
\end{aligned}$$

So $\mathbf{s}(t) = [t-1, t+2]$ *is injective. Let's consider now* $\mathbf{s}(t) = [\cos(t), 1]$*. Now if we choose* $t_1 = 2\pi$ *and* $t_2 = 4\pi$*, then*

$$\begin{aligned}
\mathbf{s}(t_1) &= \mathbf{s}(t_2) \\
\mathbf{s}(2\pi) &= \mathbf{s}(4\pi) \\
[\cos(2\pi), 1] &= [\cos(4\pi), 1] \\
[1, 1] &= [1, 1].
\end{aligned}$$

Now we get same value with two different values $t_1 = 2\pi$ *and* $t_2 = 4\pi$*. This means that* $\mathbf{s}(t) = [\cos(t), 1]$ *is not injective.*

Next we take one example of calculating the length of the curve.

Example 2.7. *Let* $s(t) = [t^2 + t, t - 1]$ *be injective parametrization of the curve* S*, such that* $t \in [0, 5]$*. Now the length can be calculated as follows*

$$l(S) = \int_0^5 \sqrt{(x'(t))^2 + (y'(t))^2}dt.$$

Let's first calculate derivatives

$$\begin{aligned}
x'(t) &= d(t^2 + t) \\
&= 2t + 1
\end{aligned}$$

$$\begin{aligned}
y'(t) &= d(t - 1) \\
&= t.
\end{aligned}$$

Now the length is

$$\begin{aligned} l(S) &= \int_0^5 \sqrt{(x'(t))^2 + (y'(t))^2}dt \\ &= \int_0^5 \sqrt{(2t+1)^2 + (t)^2}dt \\ &= \int_0^5 \sqrt{4t^2 + 4t + 1 + t^2}dt \\ &= \int_0^5 \sqrt{5t^2 + 4t + 1}dt \\ &\approx 32.538. \end{aligned}$$

Calculating the length of the curve is more accurate and more light weight way to calculate curve distance than the way which was introduced earlier. However in many cases calculating length is much more complicated to implement, because before calculation you have to be sure that parametrization is injective and derivatives of parametrizations exist. In spline cases length can be calculated in sum of spline parts lengths. [5]

2.6 Derivative of B-spline

Differentiability of the spline is important property, because we need it for defining curvature of the spline and also for defining current slope of the spline. We will start with the first derivative of the basis function.

Theorem 2.10. *Let $T = [t_0, t_1, \ldots, t_m]$ be the knot sequence, then the derivative of a basis funcation is*

$$N'_{i,k}(t) = \frac{k}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{k}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t). \tag{2.22}$$

Proof. We will follow the proof from page 59 in [20]. Let's prove this by induction. First if $k = 2$, then $N_{i,1}(t)$ and $N_{i+1,1}(t)$ are 0 or 1. Now derivation of $N_{i,2}(t)$ derivative with respect to t can be calculated as follows

$$DN_{i,2}(t) = D(\frac{t - t_i}{t_{i+1} - t_i} N_{i,1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,1}(t)).$$

Now this is either

$$\begin{aligned} DN_{i,2}(t) &= D(\frac{t}{t_{i+1} - t_i} - \frac{t_i}{t_{i+1} - t_i}) \\ N'_{i,2}(t) &= \frac{1}{t_{i+1} - t_i} \end{aligned}$$

or

$$DN_{i,2}(t) = D(\frac{t_{i+k}}{t_{i+k} - t_{i+1}} - \frac{t}{t_{i+k} - t_{i+1}})$$
$$N'_{i,2}(t) = \frac{-1}{t_{i+k} - t_{i+1}}.$$

Now we assume that this holds also for $k-1$ case, where $k > 2$. Next we will prove this for the case k. In proof we will use derivation product rule $(fg)' = f'g + fg'$. Now derivation of $N_{i,k}(t)$ derivative with respect to t can be calculated as follows

$$D(N_{i,k}(t)) = D(\frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t))$$
$$\begin{aligned} N'_{i,k}(t) = &\frac{1}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t - t_i}{t_{i+k-1} - t_i} N'_{i,k-1}(t) \\ &- \frac{1}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N'_{i+1,k-1}(t). \end{aligned}$$

By assumption that equation (2.22) holds for $k-1$ case, then we can use this formula for the $N'_{i,k-1}(t)$ and $N'_{i+1,k-1}(t)$. Then we have that

$$\begin{aligned} N'_{i,k}(t) = &\frac{1}{t_{i+k-1} - t_i} N_{i,k-1}(t) \\ &+ \frac{t - t_i}{t_{i+k-1} - t_i} \left[\frac{k-1}{t_{i+k-2} - t_i} N_{i,k-2}(t) - \frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) \right] \\ &- \frac{1}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \\ &+ \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \left[\frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) - \frac{k-1}{t_{i+k} - t_{i+2}} N_{i+2,k-2}(t) \right] \\ = &\frac{1}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{1}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \\ &+ \frac{t - t_i}{t_{i+k-1} - t_i} \left[\frac{k-1}{t_{i+k-2} - t_i} N_{i,k-2}(t) - \frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) \right] \\ &+ \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \left[\frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) - \frac{k-1}{t_{i+k} - t_{i+2}} N_{i+2,k-2}(t) \right] \\ = &\frac{1}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{1}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \\ &+ \frac{t - t_i}{t_{i+k-1} - t_i} \frac{k-1}{t_{i+k-2} - t_i} N_{i,k-2}(t) - \frac{t - t_i}{t_{i+k-1} - t_i} \frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) \\ &+ \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \frac{k-1}{t_{i+k-1} - t_{i+1}} N_{i+1,k-2}(t) - \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \frac{k-1}{t_{i+k} - t_{i+2}} N_{i+2,k-2}(t) \\ = &\frac{1}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{1}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t) \\ &+ \frac{t - t_i}{t_{i+k-1} - t_i} \frac{k-1}{t_{i+k-2} - t_i} N_{i,k-2}(t) - \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \frac{k-1}{t_{i+k} - t_{i+2}} N_{i+2,k-2}(t) \\ &+ \frac{k-1}{t_{i+k-1} - t_{i+1}} \left(- \frac{t - t_i}{t_{i+k-1} - t_i} + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \right) N_{i+1,k-2}(t). \end{aligned}$$

Now we notice that

$$\begin{aligned}
-\frac{t-t_i}{t_{i+k-1}-t_i}+\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} &= 1-\frac{t-t_i}{t_{i+k-1}-t_i}+\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}-1 \\
&= \frac{t_{i+k-1}-t_i}{t_{i+k-1}-t_i}-\frac{t-t_i}{t_{i+k-1}-t_i}+\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}-\frac{t_{i+k}-t_{i+1}}{t_{i+k}-t_{i+1}} \\
&= \frac{t_{i+k-1}-t_i-t+t_i}{t_{i+k-1}-t_i}+\frac{t_{i+k}-t-t_{i+k}+t_{i+1}}{t_{i+k}-t_{i+1}} \\
&= \frac{t_{i+k-1}-t}{t_{i+k-1}-t_i}+\frac{t_{i+1}-t}{t_{i+k}-t_{i+1}}.
\end{aligned}$$

By using this we have that

$$\begin{aligned}
N'_{i,k}(t) &= \frac{1}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{1}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t) \\
&\quad+\frac{t-t_i}{t_{i+k-1}-t_i}\frac{k-1}{t_{i+k-2}-t_i}N_{i,k-2}(t)-\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}\frac{k-1}{t_{i+k}-t_{i+2}}N_{i+2,k-2}(t) \\
&\quad+\frac{k-1}{t_{i+k-1}-t_{i+1}}\left(\frac{t_{i+k-1}-t}{t_{i+k-1}-t_i}+\frac{t_{i+1}-t}{t_{i+k}-t_{i+1}}\right)N_{i+1,k-2}(t) \\
&= \frac{1}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{1}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t) \\
&\quad+\frac{k-1}{t_{i+k-1}-t_i}\frac{t-t_i}{t_{i+k-2}-t_i}N_{i,k-2}(t)-\frac{t_{i+k}-t}{t_{i+k}-t_{i+1}}\frac{k-1}{t_{i+k}-t_{i+2}}N_{i+2,k-2}(t) \\
&\quad+\frac{k-1}{t_{i+k-1}-t_i}\frac{t_{i+k}-t}{t_{i+k-1}-t_{i+1}}N_{i+1,k-2}(t)-\frac{k-1}{t_{i+k-1}-t_{i+1}}\frac{t-t_{i+1}}{t_{i+k}-t_{i+1}}N_{i+1,k-2}(t) \\
&= \frac{1}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{1}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t) \\
&\quad+\frac{k-1}{t_{i+k-1}-t_i}\left(\frac{t-t_i}{t_{i+k-2}-t_i}N_{i,k-2}(t)+\frac{t_{i+k}-t}{t_{i+k-1}-t_{i+1}}N_{i+1,k-2}(t)\right) \\
&\quad-\frac{k-1}{t_{i+k}-t_{i+1}}\left(t-\frac{t_{i+1}}{t_{i+k-1}-t_{i+1}}N_{i+1,k-2}(t)+\frac{t_{i+k}-t}{t_{i+k}-t_{i+2}}N_{i+2,k-2}(t)\right) \\
&= \frac{1}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{1}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t) \\
&\quad+\frac{k-1}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{k-1}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t) \\
&= \frac{k}{t_{i+k-1}-t_i}N_{i,k-1}(t)-\frac{k}{t_{i+k}-t_{i+1}}N_{i+1,k-1}(t).
\end{aligned}$$

Now because equation (2.22) holds also for case k, then by induction it holds also for all cases. □

Next we will introduce algorithm for calculating nth derivative of the spline.

Theorem 2.11. *Let $N_{i,p}(u)$ be basis function, then kth derivative of the basis function is*

$$N_{i,p}(t)^{(k)} = p\left(\frac{N_{i,p-1}^{(k-1)}}{t_{i+p-1} - t_i} - \frac{N_{i+1,p-1}^{(k-1)}}{t_{i+p} - t_{i+1}}\right). \tag{2.23}$$

Proof. Whole proof adapt the proof from page 61 at the [20]. Let's prove this by induction. Let $N_{i,p}(u)$ be basis function.

Base case ($k = 2$): The first derivative of basis function by equation (2.22) is

$$N'_{i,k}(t) = \frac{k}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{k}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t).$$

Now if we derivative first derivative, then we get second derivative.

$$\begin{aligned} DN'_{i,k}(t) =& D(\frac{k}{t_{i+k-1} - t_i} N_{i,k-1}(t) - \frac{k}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t)) \\ &= \frac{k}{t_{i+k-1} - t_i} DN_{i,k-1}(t) - \frac{k}{t_{i+k} - t_{i+1}} DN_{i+1,k-1}(t) \\ &= k\left(\frac{DN_{i,k-1}(t)}{t_{i+k-1} - t_i} - \frac{DN_{i+1,k-1}(t)}{t_{i+k} - t_{i+1}}\right). \end{aligned}$$

This holds for the second derivative so now we assume that this holds also for the $k - 1$ case.

Step case: Now the kth derivative of the basis function is

$$\begin{aligned} N_{i,p}^{k}(t) =& DN_{i,p}^{k-1}(t) \\ &= D\left(p\left(\frac{N_{i,p-1}^{(k-2)}}{t_{i+p-1} - t_i} - \frac{N_{i+1,p-1}^{(k-2)}}{t_{i+p} - t_{i+1}}\right)\right) \\ &= pD\left(\frac{N_{i,p-1}^{(k-2)}}{t_{i+p-1} - t_i} - \frac{N_{i+1,p-1}^{(k-2)}}{t_{i+p} - t_{i+1}}\right) \\ &= p\left(\frac{DN_{i,p-1}^{(k-2)}}{t_{i+p-1} - t_i} - \frac{DN_{i+1,p-1}^{(k-2)}}{t_{i+p} - t_{i+1}}\right) \\ &= p\left(\frac{N_{i,p-1}^{(k-1)}}{t_{i+p-1} - t_i} - \frac{N_{i+1,p-1}^{(k-1)}}{t_{i+p} - t_{i+1}}\right). \end{aligned}$$

Because this holds for the kth derivative, then by the induction this holds for the all cases. □

B-spline derivation can be constructed now by multiplying basis function derivatives with corresponding control points.

Definition 2.19. Let $S(t)$ be spline, then kth derivative of spline $S(t)$ is

$$S^{(k)}(t) = \sum_{i=0}^{n} N_{i,p}^{(k)}(t)\mathbf{p_i}. \tag{2.24}$$

We are going to need first and second derivative of the spline for the defining the spline's current curvature. Curvature is handled in next section and it is important property of the curves. [20]

2.7 Curvature

Curvature descripes slope of the spline curve at each point of the spline. Curvature is needed for the checking that route is valid for the AGV. Before we are going to construct the method for calculating the curvature, we are going to define the curvature.

Definition 2.20 (**Curvature** [14])**.** Let $S(t)$ be the spline. Curvature k of $S(t)$ is

$$k = \frac{d\phi}{dl(S)}. \tag{2.25}$$

This means that curvature describes the rate of change in the direction of the tangent line of the spline at that point t with respect to the length of spline $l(S)$.

Next we are going to construct the way for calculating the splines' curvature at the point t. Construction follows the proof of proposition 2.1.2 at the [21]. Let S be a spline in $\mathbb{R}^2$ such that $S(t) = S_x(t)\mathbf{i} + S_y(t)\mathbf{j}$. Let also $l(S)$ be the length of the spline and t be a parameter for the spline S. Using the chain rule for splines' derivative, we can write

$$S' = \frac{dS}{dt} = \frac{dS}{dl(S)}\frac{dl(S)}{dt}.$$

For length of spline we can use theorem (2.17), when we have

$$\begin{aligned}
\frac{dl(S)}{dt} &= \frac{d}{dt}\int_{t_0}^{t} ||S'(u)||du \\
\frac{dl(S)}{dt} &= ||S'(t)|| \\
\iff \left(\frac{dl(S)}{dt}\right)^2 &= ||S'(t)||^2 \\
\iff \left(\frac{dl(S)}{dt}\right)^2 &= S' \cdot S'.
\end{aligned}$$

Next we derivate both sides respect to t, when we have

$$\begin{aligned}
\frac{d}{dt}(S' \cdot S') &= \frac{d}{dt}\left(\frac{dl(S)}{dt}\right)^2 \\
\frac{d}{dt}S' \cdot S' + S' \cdot \frac{d}{dt}S' &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
S'' \cdot S' + S' \cdot S'' &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
S'' \cdot S' + S'' \cdot S' &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
||S''||||S'||\cos\phi + ||S''||||S'||\cos\phi &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
2||S''||||S'||\cos\phi &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
2(S'' \cdot S') &= 2\left(\frac{dl(S)}{dt}\right)\frac{d^2l(S)}{dt^2} \\
S' \cdot S'' &= \frac{dl(S)}{dt}\frac{d^2l(S)}{dt^2}.
\end{aligned}$$

Now for the curvature we can write that

$$\begin{aligned}
k &= \left|\left| \frac{d^2S}{dl(S)^2} \right|\right| \\
&= \left|\left| \frac{d}{dl(S)} \left(\frac{dS}{dl(S)} \right) \right|\right| \\
&= \left|\left| \frac{d}{dl(S)} \left(\frac{\frac{dS}{dt}}{\frac{dl(S)}{dt}} \right) \right|\right| \\
&= \left|\left| \frac{\frac{d}{dt}\left(\frac{\frac{dS}{dt}}{\frac{dl(S)}{dt}}\right)}{\frac{dl(S)}{dt}} \right|\right| \\
&= \left|\left| \frac{\frac{\frac{d^2S}{dt^2}\frac{dl(S)}{dt} - \frac{d^2l(S)}{dt^2}\frac{dS}{dt}}{(\frac{dl(S)}{dt})^2}}{\frac{dl(S)}{dt}} \right|\right| \\
&= \left|\left| \frac{\frac{dl(S)}{dt}\frac{d^2S}{dt^2} - \frac{d^2l(S)}{dt^2}\frac{dS}{dt}}{(dl(S)/dt)^3} \right|\right| \\
&= \left|\left| \frac{\frac{dl(S)}{dt}S'' - \frac{d^2l(S)}{dt^2}S'}{(dl(S)/dt)^3} \right|\right| \\
&= \left|\left| \frac{(\frac{dl(S)}{dt})^2 S'' - \frac{dl(S)}{dt}\frac{d^2l(S)}{dt^2}S'}{(dl(S)/dt)^4} \right|\right| \\
&= \left|\left| \frac{||S'||S'' - (S' \cdot S'')S'}{(dl(S)/dt)^2(dl(S)/dt)^2} \right|\right| \\
&= \left|\left| \frac{(S' \cdot S')S'' - (S' \cdot S'')S'}{||S'||^2||S'||^2} \right|\right| \\
&= \frac{||(S'' \cdot S')S'' - (S' \cdot S'')S'||}{||||S'||^2||S'||^2||} \\
&= \frac{||(S' \cdot S')S'' - (S' \cdot S'')S'||}{||S'||^4}.
\end{aligned}$$

Next, we will need vector product identity ($S' \times (S'' \times S') = (S' \cdot S')S'' - (S' \cdot S'')S'$) and by using this we have

$$\begin{aligned}
k &= \frac{||(S' \cdot S')S'' - (S' \cdot S'')S'||}{||S'||^4} \\
&= \frac{||S' \times (S'' \times S')||}{||S'||^4}.
\end{aligned}$$

Because S' and $S'' \times S'$ are perpendicular, then

$$\begin{aligned}
k &= \frac{||S' \times (S'' \times S')||}{||S'||^4} \\
&= \frac{||S'||||S'' \times S'||}{||S'||^4} \\
&= \frac{||S'' \times S'||}{||S'||^3}.
\end{aligned}$$

Because spline $S(t)$ is form $S(t) = S_x(t)\mathbf{i} + S_y(t)\mathbf{j}$, then for curvature we have that

$$\begin{aligned} k &= \frac{||S'' \times S'||}{||S'||^3} \\ &= \frac{||(S''_x\mathbf{i} + S''_y\mathbf{j}) \times (S'_x\mathbf{i} + S'_y\mathbf{j})||}{||(S'_x\mathbf{i} + S'_y\mathbf{j})||^3} \\ &= \frac{||(S''_y S'_x - S''_x S'_y)||}{\sqrt{S'^2_x + S'^2_y}^3} \\ &= \frac{S'_x S''_y - S'_y S''_x}{(S'^2_x + S'^2_y)^{\frac{3}{2}}}. \end{aligned}$$

From this we can write corollary.

Lemma 2.12. *Let S be spline such that $S(t) = S_x(t)\mathbf{i} + S_y(t)\mathbf{j}$. Now the curvature of the spline at point t is*

$$k = \frac{S'_x S''_y - S'_y S''_x}{(S'^2_x + S'^2_y)^{\frac{3}{2}}}. \tag{2.26}$$

Now we have all the necessary properties of the B-splines. In next chapter we will introduce genetic algorithm which will be applied in optimization algorithm.

3 GENETIC ALGORITHM

In this section, we will introduce a genetic algorithm at the general level and we will give one concrete example of genetic algorithm usage. In the beginning, evolution algorithms were developed to simulate nature's evolution in software programs. The genetic algorithm is one of the evolution algorithms and it tries to simulate nature's genetic functionality. The genetic algorithm can be applied to mathematical problems that are hard to solve or even impossible to solve by using general mathematical methods. This algorithm doesn't find necessarily the best solution to the problem but it gives good enough approximation of the exact solution.

3.1 Description

Before going into genetic algorithms' functionality, we will define a few properties and methods for genetic algorithms. **Candidate** is the one possible solution for the given problem. It is not necessarily the best solution for the problem, but it is one kind of solution. **Population** is the group of the candidates which forms the generation and is used for finding the best solution for the problem. **Generation** represents the population "number" during the algorithm. Every new population is the new generation and when a new population is fully generated then the new population is the new generation. The first population is the first generation. **Parent** is the one candidate from the population, which is used for creating the new candidate for the next generation. **Chromosome** is the place where every candidate's genetic data is stored. This means that the chromosome has the candidates' genetic data and it defines the whole nature of the candidate. **Gene** is the genetic data of the candidate, which is stored in a chromosome. Gene defines candidates smaller properties. **Crossover** is the genetic operation which combines at-least two parents to create new candidate to the population. In most cases, the crossover method combines parents' genes to construct a new chromosome to the new candidate. **Mutation** is also the genetic operation that randomly modifies the created candidate to create more variety into the population. In most cases, the mutation affects new candidates' specific genes and changes them a little bit. [28]

Generally genetic algorithms suit best to permutation problems, but they can also be used to other kinds of mathematical problems. The key thing when constructing the genetic algorithm is to find a good enough fitness function for the algorithm. Fitness functions

give fitness value for every candidate of the population and candidates are ordered by the fitness value. Fitness value describes candidates "goodness". Another required thing for genetic algorithms is a crossover process. That means the process of how new children are formulated into the new populations. [28]

The next we will go through the genetic algorithm process and explain how it works. The following figure is an activity diagram of genetic algorithm functionality. The Activity diagram describes a basic genetic algorithm process.

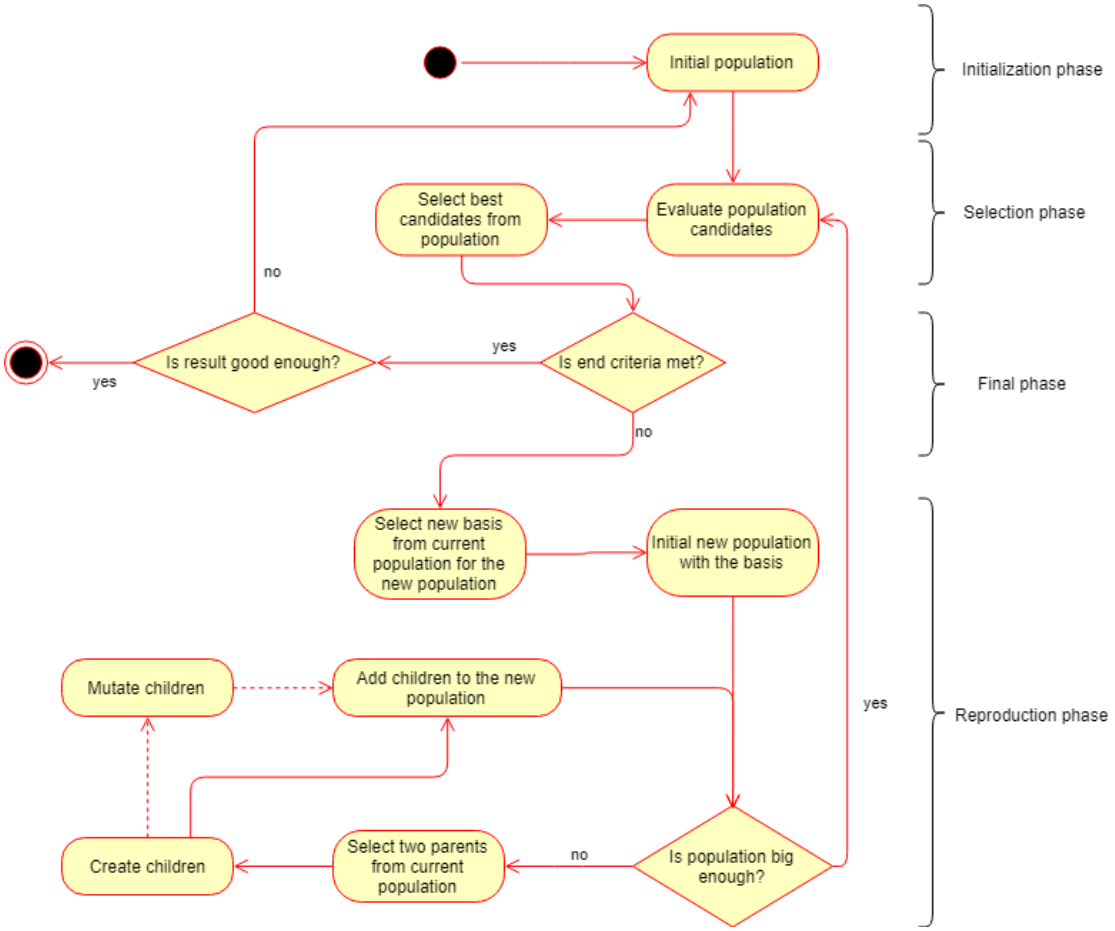


Figure 3.1. *General evolution algorithm process.*

At the beginning of the algorithm, we create the start population. The start population size is based on the population size parameter N. The first population is initialized by taking points evenly distributed across the whole point space of the problem. Sometimes, when we know which part of the point space contains the optimal solution, then we can weight that part of the space more to get the optimal result faster. Still, it would be good to take points from other places of the space too to get more variety to the population and by the variety, we can find global maximum much more likely. After population initialization, all the members of the population are evaluated by a fitness function. Evaluation gives fitness value for all members of the population and by fitness value, all the members can be ordered in ascendant or descent order depending on the nature of fitness value. [28]

After ordering the values we select the best candidates to form a basis for the new population. The amount of best candidates are selected depends on the basis size variable N_b. The new population is smaller than the original population so we will need to form children from two randomly picked parents to grow population size. The number of children depends on the size of the original population and the new population basis size. Number of children can be calculated by a formula

$$N_c = N - N_b. \tag{3.1}$$

Children are formulated by using parents. A parent is selected such that we pick randomly candidate from a previous population such that we favor better candidates. The probability of the better candidate is going to be selected depends on parent favor parameter $p_{betterfavour}$. Children are constructed by the crossover method, which combines both parents' genetical properties. [28]

Children are also randomly mutated, children mutation depends on the mutation probability parameter which determines how likely children are mutated. Mutation effects on children's properties and it creates more variety to the population. After enough children are made and the population is big enough, then the new population is evaluated again, the best candidates are selected and new population creation starts again. After every population evaluation, algorithm end criteria are checked and if at least one ending criterion is met, then the algorithm stops. [28] The Next figure illustrates child creation functionality and how it affects population candidates' overall fitness value.

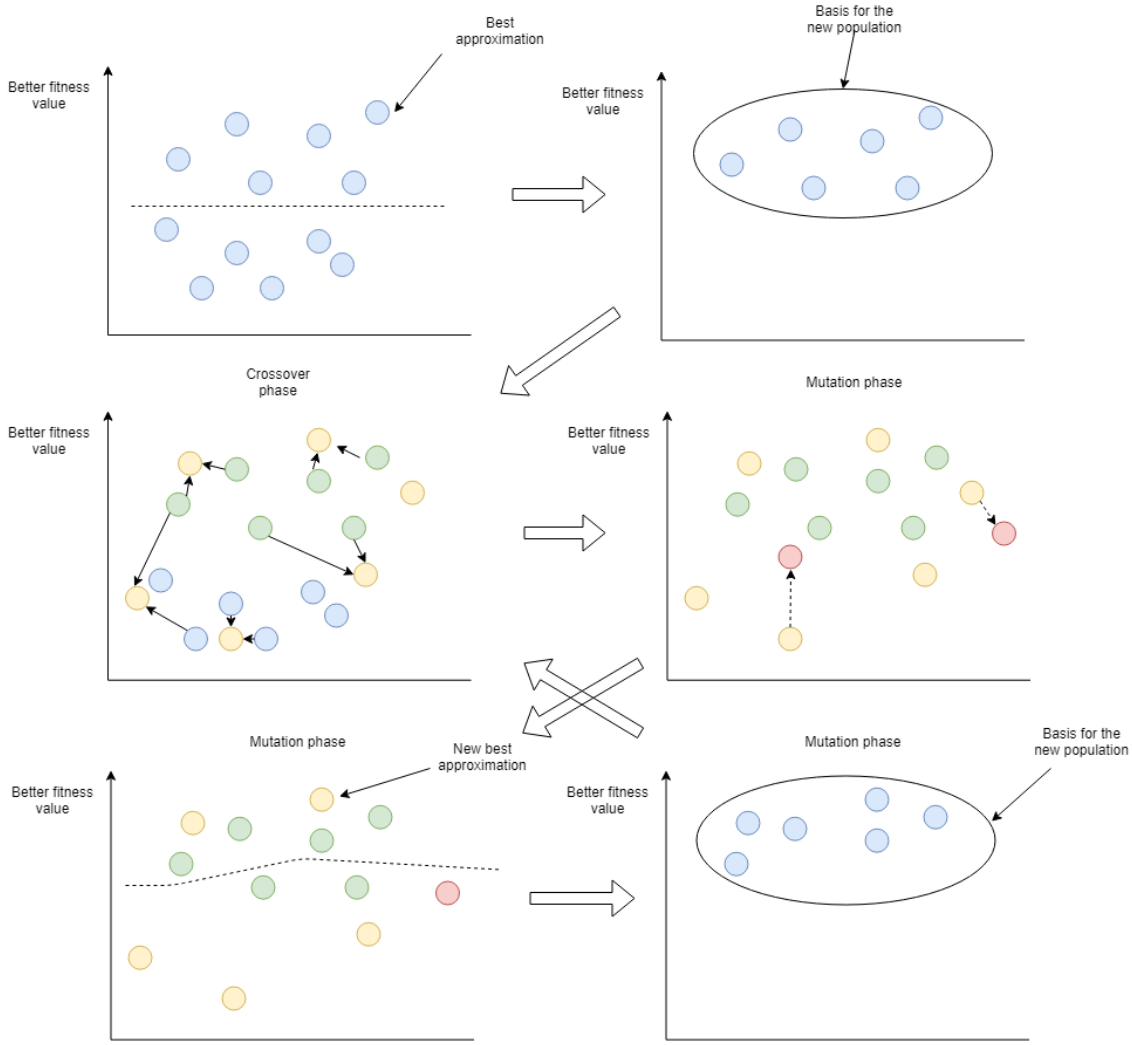


Figure 3.2. *Creating a new population using an evolution algorithm.*

As we can see from the above figure the best fitness value of the population gets always better or stays at least the same. This leads to another problem. Let's consider a scenario where fitness function has one local maximum which is lower than the global maximum of the function. Now if these points are separated enough from each other, then the genetic algorithm can't always find global maximum and it will then only find local maximum. This happens because the best fitness value of the population is always bigger or equal to the previous one. This situation is illustrated in the next figure.

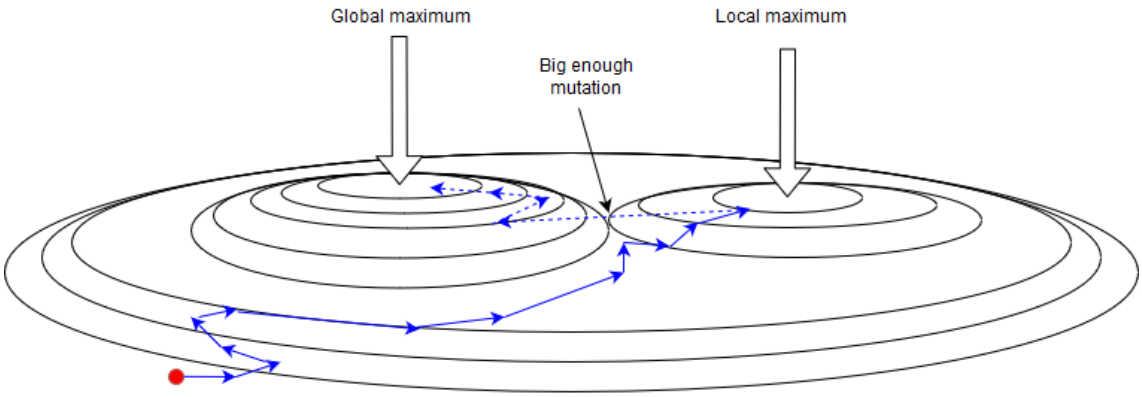


Figure 3.3. *Finding the maximum fitness value using a genetic algorithm.*

Now if those two separate tops in the previous figure are close enough each other, then it might be possible that we can reach global maximum, but in most cases, we can only find local maximum. That's why the genetic algorithm always finds a local maximum of the fitness function in the given space and we can't say that the given maximum is the global maximum of the function. [28]

3.2 Example "salesman problem"

The salesman problem is a widely known permutation problem where the goal is to find the shortest path between the different amount of cities. This is a good example of the automated truck system because this could also be applied for selecting the shortest path between the different amounts of endpoints. Of course, that problem is not as simple as this example, but by modifying fitness function, this could also be applied to that system.

Each traveled point is numbered sequentially so that we can define the traveled route explicitly. After that, candidates are constructed such that every candidate contains all the cities once. The candidate represents the traveled route. In this algorithm fitness function is the overall distance of the path. Let's assume that we have a k-number of points where the salesman has to go and let's assume also that we have n-dimensional space. In real life situation, we would consider in most of the cases 2-dimensional case, but this can be applied also into n-dimensions. [28] Because salesman have to come to the home after the tour we need to take account also the distance from the last point to the first point so we have the following fitness function

$$f(\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_k}) = \sum_{i=1}^{k-1} \left(\sqrt{\sum_{j=1}^{n} (x_{i_j} - x_{i+1_j})^2} \right) + \sqrt{\sum_{j=1}^{n} (x_{k_j} - x_{1_j})^2} \tag{3.2}$$

Children's formulation is made such that we pick two parents and combine their properties. We take the first part from parent 1 and the second part from parent 2 and then we combine two different children using those parts. After both children are created, we can have one or more points appear at route multiple times. This can be corrected by

swapping those cities between children. [28] This children inheritance is described more detailed in the following figure.

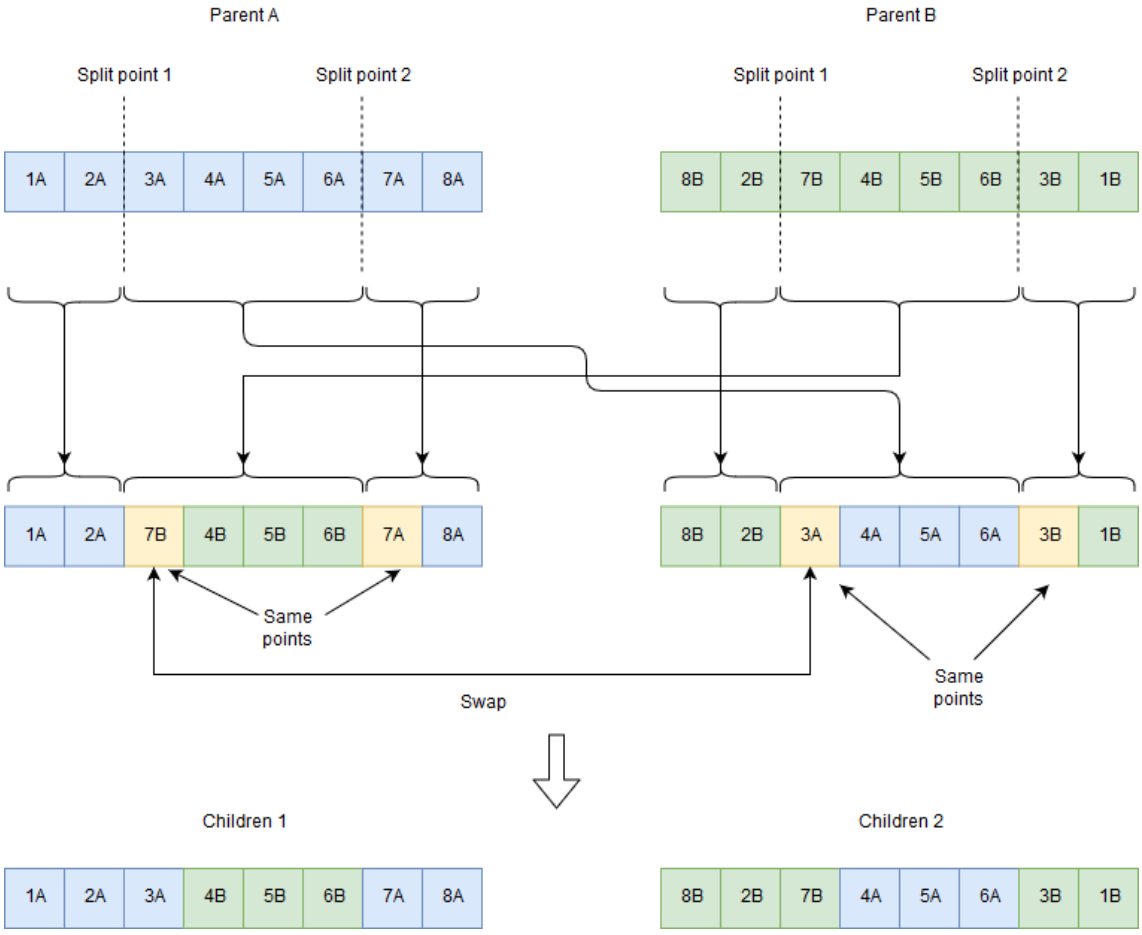


Figure 3.4. *Salesman problem children inheritance.*

Mutating children is made by swapping one point pair places in sequence [28]. The next figure will describe the mutation process more detailed.

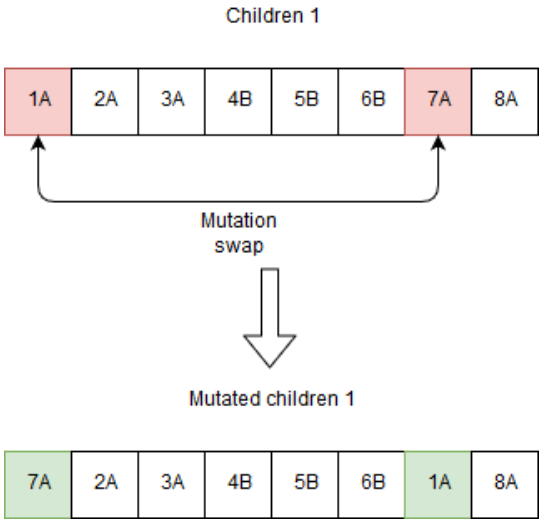
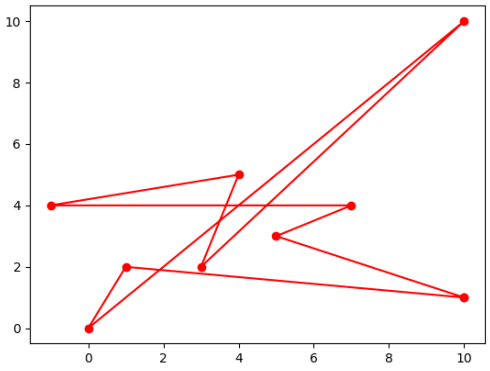
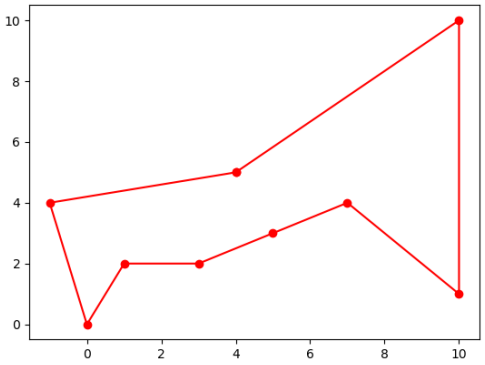


Figure 3.5. *Salesman problem children mutation.*

The next figures show how this example genetic algorithm works for nine arbitrary points.



(a) *Starting route of example usage.*



(b) *The best route of example usage, resolved with genetic algorithm.*

Figure 3.6. *Routes before and after salesman problem genetic algorithm.*

Traveled distance in the starting route is 31.32 and after genetic algorithm distance is 22.17, so traveled distance shortened about 29%.

3.3 Why genetic algorithm works?

The genetic algorithm is a heuristic method, so it is not based on mathematical rules and methods. The genetic algorithm is inspired by nature's evolution algorithms and it tries to simulate nature's genetic functionalities. Many genetic algorithm operations and variables names come from natural functionalities. One way to explain why genetic algorithms work is Holland's schema theorem. [28] Before we can go into that, we need to define the schema.

Definition 3.1. Schema is a template that defines a subset. The subset includes chromosomes with the same sections, which are defined in the template.

Let's say that we have a genetic algorithm that tries to find the three number password and we know two numbers of the password. Let * be called a wild number. Now the one possible schema could be $(*23)$. Then the subset would be $\{(123), (223), (323), (423), ..., (923)\}$ which fulfills the schema. [23]

One commonly used theorem for genetic algorithms theoretical analysis is Holland's schema theorem. This theorem has its own limitations, but this can be used in this case for explaining why genetic algorithms work. [29] For proving the Holland's schema theorem, we need to define few parameters. At first, let S be the schema, t to be the generation and $P(t)$ to be the population. The first important parameter is fitness ratio

$$r(S,t) = \frac{f(S,t)}{\overline{f}(t)}. \tag{3.3}$$

Here $f(S,t)$ is the fitness value of the schema, which is the average fitness value of the all-candidates' fitness values that match the schema, and it can be calculated by

$$f(S,t) = \frac{\sum_{x \in S \cap P(t)} f(x)}{|S \cap P(t)|}. \tag{3.4}$$

Respectively $\overline{f}(t)$ is the average fitness value of the generation t and it can be respectively calculated by

$$\overline{f}(t) = \sum_{x \in P(t)} \frac{f(x)}{|P(t)|}. \tag{3.5}$$

One important parameter is also a number of instances of a schema S at the generation t. Let's mark this by $N(S,t)$. This represents the number of instances that fulfills the schema in the generation t. [23] Before constructing the Holland's schema theorem we go through few smaller Lemmas.

Lemma 3.1. *Let $N(S,t)$ be the number of instances of schema S at the generation t. Let parent selection be made in proportion to fitness values. Now the expected number of instances of a schema S at the generation $t+1$ is*

$$E[N(S,t+1)] = r(S,t)N(S,t). \tag{3.6}$$

Proof. The proof adapts the proof in [9]. For expected value we know that

$$E[X] = \sum_{i=1}^{k} x_i p_i.$$

Now because parent is selected by the fitness-proportional selection, then probability that parent will be selected is

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}.$$

Candidate can be selected multiple times from the population and from statics we know that number of times candidate is selected is directly proportional to $\frac{f(x)}{\overline{f}(t)}$, where $f(x)$ is fitness value of candidate. Because of that, expected number of instances of a schema

S at the generation $t+1$ is

$$\begin{aligned} E[N(S,t+1)] &= \sum_{x \in S} \frac{f(x)}{\overline{f}(t)} \\ &= \frac{1}{\overline{f}(t)} \frac{N(S,t)}{N(S,t)} \sum_{x \in S} f(x) \\ &= \frac{1}{\overline{f}(t)} N(S,t) \sum_{x \in S} \frac{f(x)}{N(S,t)} \\ &= \frac{f(S,t)}{\overline{f}(t)} N(S,t) \\ &= r(S,t) N(S,t). \end{aligned}$$ □

The previous lemma only considers parent selection and it doesn't take account any kind of mutation or crossover. In genetic algorithm, mutation or crossover can change the schema of the candidate which affects to expected number of instances of S at generation t. Next, we will consider the probability that schema S is represented in the population after crossover.

Lemma 3.2. *Let p_c be the crossover probability to a schema S, t be the generation and $l(S)$ to be the length of schema S. Also, let l be the length of the chromosome and $P_{diff}(S,t)$ be the probability of the other parent being an instance of a different schema S'. Then*

$$1 - p_c \frac{l(S)}{l-1} P_{diff}(S,t) \tag{3.7}$$

is the lower bound of the probability of schema S being represented at generation $t+1$ population.

Proof. Proof adapts the proof of Lemma 2.3 in [23]. The first we will consider the probability of the schema being destroyed by the crossover method. Let the crossover probability be p_c. The probability of the schema being destroyed by the crossover method is the product of the crossover probability and the probability of the crossover place being inside the length of the schema. The probability of the crossover place being inside the length of the schema is the length of the schema divided by the length of chromosome minus one $\frac{l(S)}{l-1}$. So now the probability of the schema being destroyed is

$$p_c \frac{l(S)}{l-1}. \tag{3.8}$$

Because we select at-least two parents for the new candidate, then the other parent may be an instance of S. If the other parent is the instance of S, then S will survive no-matter the crossover place. Because of that, we need to multiply our probability with the probability of the other parent being the instance of a different schema. So now the probability of the schema being destroyed is $p_c \frac{l(S)}{l-1} P_{diff}(S,t)$. Now by taking the

complement of that probability we get the probability for the non-destruction, which is now

$$1 - p_c \frac{l(S)}{l-1} P_{diff}(S,t). \tag{3.9}$$

S may appear in the generation $t+1$ by the result of crossover from two different schemata. Because of that probability for the non-destruction is a lower bound. □

Now we have the lower bound for the probability that schema S is represented in the population after crossover. Next, we will consider the probability that schema S is represented in the population after mutation.

Lemma 3.3. *Let t be the generation, p_m be the mutation probability per gene, S be a schema and $k(S)$ be the order of the schema. Then*

$$1 - p_m k(S) \tag{3.10}$$

is the lower bound of the probability of S being represented in the population at generation $t+1$.

Proof. Proof adapts the proof of Lemma 3.3 in [23]. The probability of mutation happening in one gene is p_m, so the probability of mutation not happening is $1 - p_m$. Now the probability of mutation not happening to the schema is $(1-p_m)^{k(S)}$. Now by approximating that we get lower bound for the probability.

$$\begin{aligned}(1-p_m)^{k(S)} &= \sum_{i=0}^{k(S)} \binom{k(S)}{i} (-p_m)^k \\ &= 1 - k(S)p_m + \sum_{i=2}^{k(S)} \binom{k(S)}{i} (-p_m)^k \\ &\geq 1 - p_m k(S).\end{aligned}$$

□

Now when we have the lower bound for the probability that schema S is represented in the population after crossover and after the mutation, then by combining both of these with lemma (3.1) we get the Hollands' Schema Theorem.

Theorem 3.4 (The Hollands' Schema Theorem)**.** *Let p_m be mutation probability and p_c be crossover probability. Let also S be the schema. Now the expected number of representatives of schema S at the generation $t+1$ is*

$$E(N(S,t+1)) \geq (1 - p_c \frac{l(S)}{l-1} P_{diff}(S,t) - p_m k(S)) r(S,t) N(S,t). \tag{3.11}$$

Theorem (3.4) says that above average fitness value, short defining length and lower order schemas are going to last longer through the generations. Of course, this theorem

assumes certain things and is not valid for all cases, but it is one way to explain why genetic algorithms work in certain cases. There are other explanations also to that question, but we won't go through them. [23]

4 SYSTEM DESCRIPTION

This optimization problem has many properties and restrictions that have to be taken into account. In this section, we will go through all the initial conditions and restrictions of the problem. We also introduce the ways for checking those restrictions at the algorithm, but the first we will tell shortly about the AGVs and automated warehouses.

4.1 Automated guided vehicles and automated warehouse

Automated guided vehicles are automatically moving vehicles, which are designed to handle routine goods transports in the warehouse. AGV's goal is to increase efficiency and improve production performance. AGV's can operate without a driver because it is automatically navigated. [18] There are a few possible ways to handle AGV's navigation. One possible way is laser triangulation. In this method, the AGV position is recognized by using the laser sensor. AGV location can be detected in realtime and by that AGV can track the given trajectory. Usually, a laser sensor is attached to the top of the AGV and it is connected to the AGV's control system. This method needs a good mapping of obstacles and the whole warehouse. [4][10]

The other method is floor wiring, where AGV navigation is handled by wires on the floor, which AGV follows. Wires are followed by the sensor in AGV. This method doesn't require such careful and detailed mapping of the warehouse, because vehicles are only following the tapes and moving along them. [10]

The automated warehouse system has multiple different tools, for example, a control tool which controls AGV's tasks and route optimizing tool, which controls AGV's routes and selects the best route for the current task. One of the tools is the warehouse planning tool, where all the possible routes are planned and defined. This work is meant to the planning tool for optimizing the planned routes. [18][1]

4.2 System restrictions and initial conditions

This section's purpose is to introduce all the optimization restrictions. We introduce the most important restrictions from an optimization point of view so there might be some other restrictions too which we don't introduce. But first, we need to describe the system we are dealing with. The following figure illustrates the optimization situation.

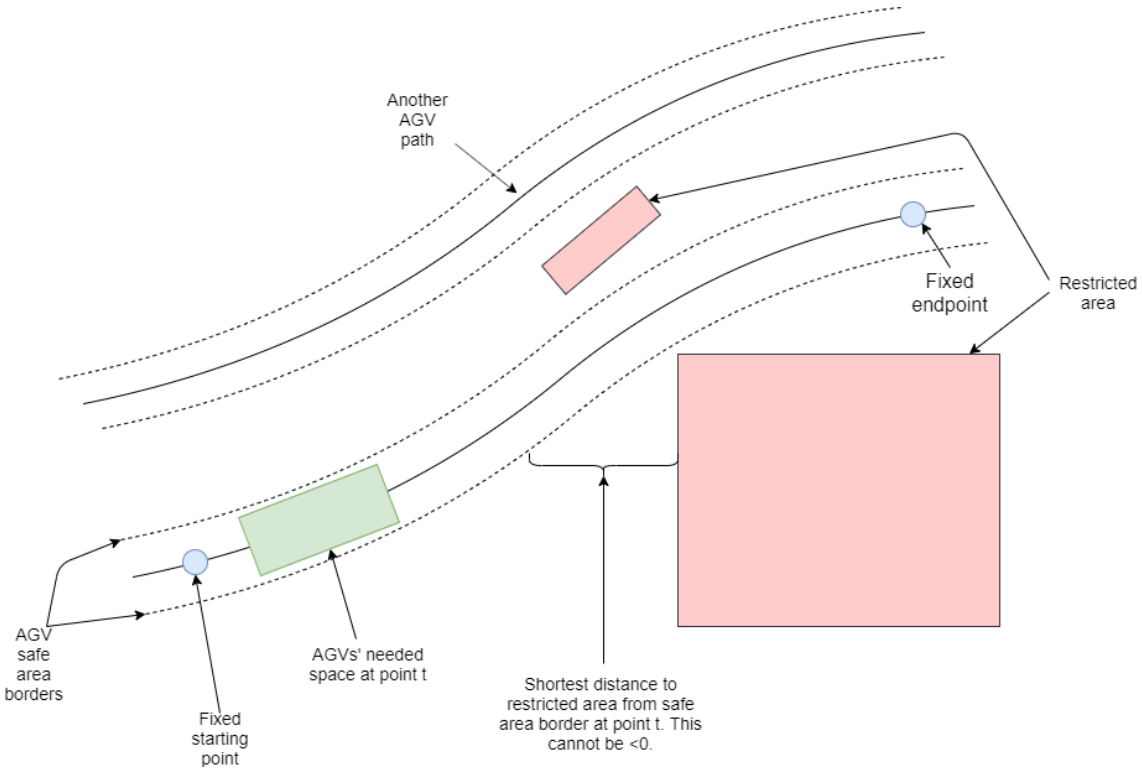


Figure 4.1. *Illustration of optimization situation.*

Next, we are going through the components of the system and other components that affect optimization. The AGV safe area defines the area AGV needs while driving the route. Here safe area borders are at the same distance from the AGV route at all points of the route. Of course, in the real world, it is not constant, because AGV is not in the same angel respect to the path in every point of the route. In this research, we define the safe area to be so big that the whole AGV stays inside the safe area at all points of the route. This can be done because it doesn't affect this research result because we are researching that if the genetic algorithm suits this kind of problem. We develop the algorithm so that this can be easily taken into account without needing to change much in the algorithm. We are going to talk more about this later when we are talking about algorithm development proposals. In algorithm development proposals, we will also talk about other AGV paths and how to take them into account. In this research, we will handle other AGV paths as obstacles, because it doesn't affect the result of this research and it simplifies the algorithm quite a bit.

The next we will go into restrictions and initial conditions of the optimization situation. In optimization all the suitable route candidates have to fulfill all the following conditions:

- AGV has acceleration at the starting point a_0.
- AGV has starting velocity v_0.
- Wheel turning angle can't exceed AGV's tire's maximum turning angle ϕ_{max}.
- Wheel turning velocity can't exceed defined limit value.
- Velocity of the AGV cannot exceed maximum velocity of the AGV v_{max}.

- Acceleration cannot exceed AGV's maximum acceleration or deceleration.
- Centripetal acceleration cannot exceed it's limits in different curves, because AGV has to be able to drive the curve.
- Curve has to be continuous at the starting point and end point.
- Curve should be derivable (smooth) at the starting and end point.
- Start and end point are fixed, they are not allowed to change.
- Start and end point angle can't change.
- Path cannot go through the restricted area.
- Two paths cannot cross each other.
- Safe area of the AGV cannot touch restricted area.

The continuous curve at the start and endpoint of the path ensures that the spline's start point and endpoint remain attached to another route. Differentiability at the starting point and endpoint ensures that there will not be any spikes in the starting and ending of the optimized curve. This means that the AGV won't need to stop at the start and endpoint of the optimized curve.

We also need to assume that the AGV has starting velocity v_0 at the beginning of the curve t_0. Assume also that the AGV has starting acceleration a_0. If $a_0 = 0$, then the AGV is moving constant velocity and if $a_0 < 0$, then AGV velocity is slowing down, otherwise it's velocity is growing. From mechanics, we know that

$$a(t) = \frac{dv}{dt} = \frac{d^2x}{dt}, \tag{4.1}$$

where v is the velocity of the AGV and x is the place [17].

Let spline be $S(t)$. We know that spline is parametrized polynomial function which can be written as follows $S(t) = \mathbf{i}S_x(t) + \mathbf{j}S_y(t)$. This mean that in given point t we have vector from origo to point $S(t)$. In this representation $\mathbf{i}$ component represents x-coordinate component vector and $\mathbf{j}$ component represents y-coordinate component vector. So if we now take arbitrary point $S_i(u_i)$, then next point $S_j(u_j)$ is

$$\begin{aligned} \mathbf{S}_j(t_j) &= \mathbf{S}_i(t_j) + \mathbf{s}_{ij} \\ \mathbf{s}_{ij} &= \mathbf{S}_j(t_j) - \mathbf{S}_i(t_j). \end{aligned}$$

Now if we take derivative respect to time we get that

$$\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i. \tag{4.2}$$

If we derivative respect to time one more time, we get equation for the acceleration

$$\mathbf{a}_{ij} = \mathbf{a}_j - \mathbf{a}_i. \tag{4.3}$$

In this system $\mathbf{a}_j$ and $\mathbf{a}_i$ is restricted by $a_{max,\,acceleration}$ and $a_{max,\,decelaration}$, where $a_{max,\,acceleration}$ is maximum acceleration of the AGV and $a_{max,\,decelaration}$ is maximum deceleration of the AGV. Here $\mathbf{v}_{ij}$ is the change of the velocity between points i and j and $||\mathbf{s}_{ij}||$ is the straight distance between point i and j. Velocity is restricted to be between 0 and v_{max}, which is the maximum velocity of the AGV.

4.3 Condition checking and calculating drive time through the spline

Let's consider independent data of the system, which means that data can be calculated for each section without calculating previous section values. This kind of data is for example wheel maximum turning angle and maximum centripetal acceleration. Centripetal acceleration is proportional to curvature k_i and drive velocity. So now section's maximum allowed velocity in section j can be calculated as follows

$$v_{j,max} = \min\left(v_{max}, \sqrt{\frac{a_{maxcentripetal}}{k_j}}\right), \tag{4.4}$$

where $a_{maxcentripetal}$ is maximum centripetal acceleration and k_j is curvature of jth section of spline [12]. We need to take a minimum because AGV's velocity cannot exceed the maximum velocity of the AGV v_{max}. Steering angle at the current sample point can be calculated based on AGV length and curvature of the sample point. We know that curvature at the sample point is inversely proportional to the radius of the circle in the current sample point. With that information and basic geometry calculus, we have that

$$\begin{aligned}
\tan(\phi_i) &= \frac{l_{vehicle}}{R} \\
&= l_{vehicle}\frac{1}{R} \\
&= l_{vehicle} \cdot k_i \\
\rightarrow \phi_i &= \arctan(l_{vehicle} \cdot k_i),
\end{aligned}$$

where ϕ_i is current steering angle, $l_{vehicle}$ is AGVs' length from center of back tire to center of front tire and k_i is sample point current curvature [12]. Following figure will describe the current situation.

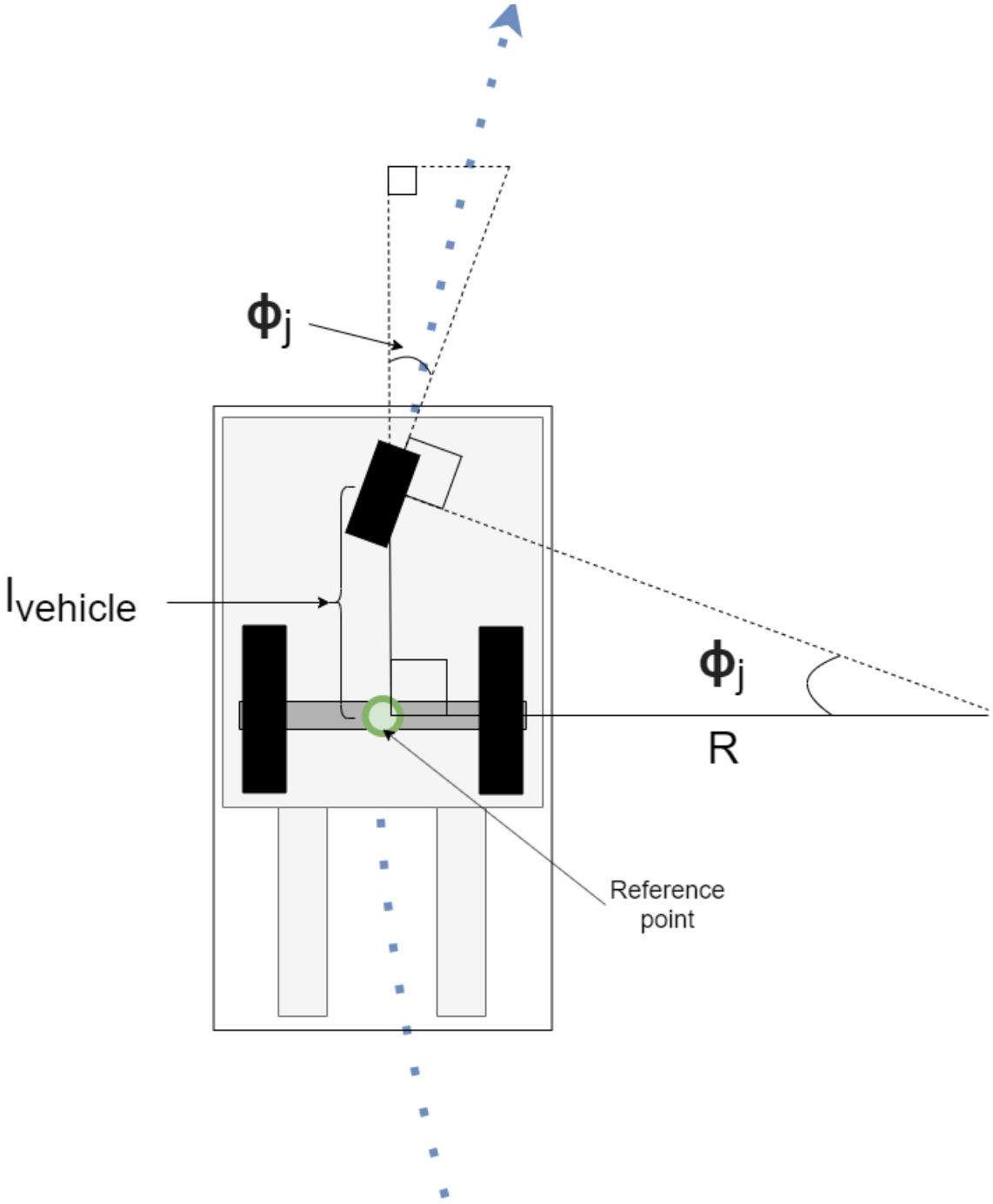


Figure 4.2. *Description of steering angle calculation.*

Next, we will consider dependent data which is data that requires information from the previous sections and is calculated based on those values. In every sample point, we need to also check that AGV doesn't collide to not allowed area or another AGV path. If we take two points from not allowed area border and those points are close enough to each other, then the route from the first point to the second point is a straight line. Because line goes through both points, it's line formula form is

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1. \tag{4.5}$$

For checking the collision in each sample point we formulate a segment through sample point which is perpendicular to the tangent line of the sample point. Segment's slope is now an opposite number of the inverse of the tangent line's slope. The tangent line's slope can be calculated by its derivate at the sample point. The segment is constructed so that it starts from a sample point and moves tangent's perpendicular direction. Segment's length is half of the vehicle's width and width of the safe area. We can formulate now line formula, for the line which goes through the sample point and it's perpendicular to the tangent line

$$\begin{aligned} y - S_y(u_j) &= -1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)}(x - S_x(t_j)) \\ y &= -1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)}(x - S_x(t_j)) + S_y(t_j). \end{aligned}$$

Because we are creating borders for the safe area we need to move both directions from the sample point. We know that distance between points can be calculated as follows

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{4.6}$$

Let $(S_x(t_j), S_y(t_j))$ be sample point of the curve. Now we can insert for y_2 that previous line equation and we get

$$\begin{aligned} \frac{w_{vehicle}}{2} + d_{safe} &= \sqrt{(x_2 - S_x(t_j))^2 + (y_2 - S_y(t_j))^2} \\ (\frac{w_{vehicle}}{2} + d_{safe})^2 &= (x_2 - S_x(t_j))^2 + (y_2 - S_y(t_j))^2 \\ (\frac{w_{vehicle}}{2} + d_{safe})^2 &= (x_2 - S_x(t_j))^2 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)}(x_2 - S_x(t_j)) + S_y(t_j) - S_y(t_j))^2 \\ (\frac{w_{vehicle}}{2} + d_{safe})^2 &= (x_2 - S_x(t_j))^2 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2(x_2 - S_x(t_j))^2 \\ (\frac{w_{vehicle}}{2} + d_{safe})^2 &= (1 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2)(x_2 - S_x(t_j))^2 \\ (x_2 - S_x(t_j))^2 &= \frac{(\frac{w_{vehicle}}{2} + d_{safe})^2}{(1 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2)} \\ \sqrt{(x_2 - S_x(t_j))^2} &= \pm\sqrt{\frac{(\frac{w_{vehicle}}{2} + d_{safe})^2}{(1 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2)}} \\ x_2 - S_x(t_j) &= \pm\frac{\sqrt{(\frac{w_{vehicle}}{2} + d_{safe})^2}}{\sqrt{(1 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2)}} \\ x_2 &= \pm\frac{\frac{w_{vehicle}}{2} + d_{safe}}{\sqrt{(1 + (-1 \cdot \frac{S'_x(t_j)}{S'_y(t_j)})^2)}} + S_x(t_j). \end{aligned}$$

So in general form x-coordinates can be calculated as follows, when we are moving to k slope and distance d

$$x_2 = \pm \frac{d}{\sqrt{1+k^2}} + x_1. \tag{4.7}$$

Now when we have x-coordinate for both endpoints, then we can calculate also y-coordinates by inserting both solved x-coordinates to line formula

$$y_2 = \frac{S'_y(t_j)}{S'_x t_j}(x_2 - S_x(t_j)) + S_y(t_j). \tag{4.8}$$

We have two different border points for the safe area border. Both points are on a different side of the AGV, so another point is the upper border point and the other is lower border point. Next, we want to check if the collision is detected in the current sample point. This is done by calculating the segment's between jth safe area border point and $j-1$th safe area border point and not allowed area borders intersection point and checking if the point is inside the safe-area border. Basically, at the first we calculate all the safe area border points for the truck and separate upper border and lower border for different sets. After that we start from the second border point and move all the way to the last border point. This is done separately for each borders. In each point we create segment between current point and previous point and find the intersection points between the not allowed area borders. If the intersection point is found inside the both segments, then crash happens. We illustrate this collision checking in the following figure.

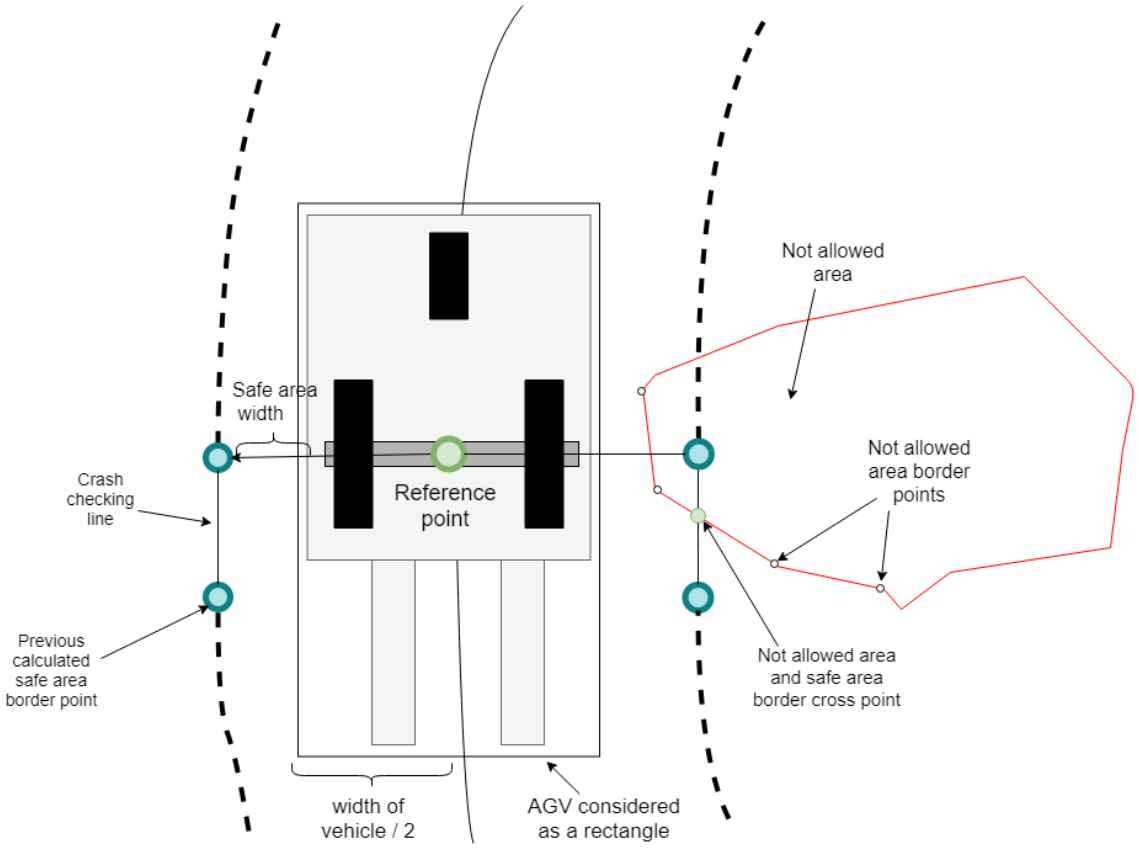


Figure 4.3. *Illustration how collision is checked.*

In two lines intersection point, y and x-coordinates are the same. Because we know the segment's line formula and can calculate line formula to the border, then we can calculate the intersection point by setting both formulas to be equal and solving x-coordinate from there. Now let ith allowed area border point be r_i and slope to be k_r and let also calculated jth safe area border be b_j and slope to be k_b. Now the slope of the segment between not allowed area border points is

$$k_r = \frac{r_{j,y} - r_{(j-1),y}}{r_{j,x} - r_{(j-1),1}}. \tag{4.9}$$

Respectively, slope of the segment between jth and $j-1$th safe area border points is

$$k_b = \frac{b_{j,y} - b_{(j-1),y}}{b_{j,x} - b_{(j-1),x}}. \tag{4.10}$$

Now, we can calculate the x-coordinate of the cross point, which is

$$\begin{aligned}
k_r(x - r_{(j-1),x}) + r_{(j-1),y} &= k_b(x - b_{j,(x-1)}) + b_{(j-1),y} \\
k_r x - k_r r_{(j-1),x} + r_{(j-1),y} &= k_b x - k_b b_{(j-1)j,x} + b_{(j-1),y} \\
k_r x - k_b x &= -k_b b_{(j-1),x} + b_{(j-1),y} + k_r r_{(j-1),x} - r_{(j-1),y} \\
(k_r - k_b)x &= -k_b b_{(j-1),x} + b_{(j-1),y} + k_r r_{(j-1),x} - r_{(j-1),y} \\
x &= \frac{-k_b b_{(j-1),x} + b_{(j-1),y} + k_r r_{(j-1),x} - r_{(j-1),y}}{k_r - k_b}.
\end{aligned}$$

Now if both line's slope is same, then they doesn't have any intersection point if

$$k_r r_{(j-1),x} + r_{(j-1),y} \neq k_b b_{(j-1),x} + b_{(j-1),y}, \tag{4.11}$$

then lines are equal. Now y-coordinate can be calculated by inserting x-coordinate to either line equation. Because we are dealing with segment and we used line formula for calculating intersection point we also need to ensure that the intersection point is in the segment. This is done by calculating distance from the intersection point to both endpoints of the segment. The border is also segment, so this needs to be also done for the border. After calculating distances to endpoints of the segments, we calculate the straight length of the segment. If the length of the segment is equal to the sum of distances from the intersection point, then the intersection point is in the segment.

Lemma 4.1. *Point is in the segment if the length of the segment is equal to length from the segment start point to point and from point to segment endpoint.*

Proof. Let $P_1 = (x_1, y_1)$ be start point of the segment and $P_2 = (x_2, y_2)$ be end point of the segment. Now let's take arbitrary point $P_3 = (x_3, y_3)$ and construct vector $\mathbf{v} = \overrightarrow{P_1 P_3}$ and other vector $\mathbf{u} = \overrightarrow{P_3 P_2}$. Now vector $\overrightarrow{P_1 P_2}$, which is same as our segment, can be represented as a sum of vectors $\mathbf{u}$ and $\mathbf{v}$ as follows $\mathbf{v} + \mathbf{u}$. Now length of the segment is same as norm $||\mathbf{v} + \mathbf{u}||$. We know that $||\mathbf{u}||^2 = u \cdot u$ and Cauchy-Schwarz inequality $\mathbf{v} \cdot \mathbf{u} \leq ||u|| \cdot ||v||$, so now

$$\begin{aligned}
||\mathbf{v} + \mathbf{u}||^2 &= (\mathbf{v} + \mathbf{u}) \cdot (\mathbf{v} + \mathbf{u}) \\
&= \mathbf{v} \cdot \mathbf{v} + \mathbf{v} \cdot \mathbf{u} + \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{u} \\
&= ||\mathbf{v}||^2 + 2\mathbf{v} \cdot \mathbf{u} + ||\mathbf{u}||^2 \\
&\leq ||\mathbf{v}||^2 + 2||\mathbf{v}|| \cdot ||\mathbf{u}|| + ||\mathbf{u}||^2 \\
&= (||\mathbf{v}|| + ||\mathbf{u}||)^2 \\
\Leftrightarrow ||\mathbf{v} + \mathbf{u}|| &\leq ||\mathbf{v}|| + ||\mathbf{u}||.
\end{aligned}$$

Here $||\mathbf{v}||$ is length from starting point to arbitrary point and $||\mathbf{u}||$ is distance from arbitrary point to end point. This means that always segment length is smaller or equal than length from starting point to arbitrary point and from there to end point. This is only equal when

Cauchy-Schwarz equation is equal, that means the case when $\mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}||||\mathbf{u}||$. We know that $\mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}||||\mathbf{u}||cos(\theta)$, where θ is vectors mutual angle. So equality is only possible when $\cos(\theta) = 1 \Leftrightarrow \theta = 0$ and that is only when vectors mutual angle is 0. If angle is 0 then vectors point in to same direction. Now because $\mathbf{v} + \mathbf{u}$ is the sequence, then angle must be 0, when $||\mathbf{v} + \mathbf{u}|| = ||\mathbf{v}|| + ||\mathbf{u}||$ and then arbitrary point is in the sequence. □

Let's next assume that inside the section, acceleration remains constant, but it can differ through different sections and also assume that at the beginning of the route we have initial velocity v_0. From the basics of the mechanics, we know that when we have constant acceleration, then velocity after time t can be calculated as follows

$$v(t) = v_0 + at, \tag{4.12}$$

where a is acceleration [17]. By solving time t from that equation we have that

$$t = \frac{v(t) - v_0}{a}. \tag{4.13}$$

We also know from mechanics that traveled distance with constant acceleration can be calculated as follows

$$x(t) - x_0 = \frac{1}{2}at^2 + v_0 t, \tag{4.14}$$

where x_0 is place at $t = 0$, t is travel time and v_0 is velocity at $t = 0$ [17]. Now we can insert solved t from above to this equation and solve velocity from that, then we have

$$\begin{aligned}
x(t) - x_0 &= \frac{1}{2}at^2 + v_0 t \\
&= \frac{1}{2}a(\frac{v(t) - v_0}{a})^2 + v_0(\frac{v(t) - v_0}{a}) \\
&= \frac{1}{2}a\frac{(v(t) - v_0)^2}{a^2} + \frac{v_0 v(t) - v_0^2}{a} \\
&= \frac{v(t)^2 - 2v(t)v_0 + v_0^2}{2a} + \frac{2v_0 v(t) - 2v_0^2}{2a} \\
&= \frac{v(t)^2 - 2v(t)v_0 + v_0^2 + 2v_0 v(t) - 2v_0^2}{2a} \\
&= \frac{v(t)^2 - v_0^2}{2a} \\
&= \frac{v(t)^2 - v_0^2}{2a} \\
\iff 2a(x(t) - x_0) &= v(t)^2 - v_0^2 \\
\iff v(t)^2 &= 2a(x(t) - x_0) + v_0^2 \\
\iff v(t) &= \sqrt{v_0^2 + 2a(x(t) - x_0)}.
\end{aligned}$$

We will go spline through 2 times. First, we will move spline forward from starting point to the endpoint and after that, we will move it back from endpoint all the way to the starting point. Forward moving considers only acceleration of the vehicle so we have to move spline backward to get also vehicles deceleration into account.

Now velocity for each section can be calculated as follows

$$v_{j,f} = \min(v_{j,max}, \sqrt{(v_{j-1,f})^2 + 2 \cdot a_{max,\, acceleration} \cdot ||s_j||}), \tag{4.15}$$

when we move the path forward from starting point to endpoint. We need to take a minimum with maximum allowed velocity in the curve because velocity cannot exceed maximum allowed velocity in current curvature.

Next, we will start from the end of the spline and come back all the way back to the starting point. We start with the final velocity of the vehicle. Then by taking minimum with the velocity of that part which we calculated when we moved forward the path, we have the vehicle velocity for the each section

$$v_j = \min(v_{j,f}, \sqrt{(v_{j+1})^2 + 2 \cdot a_{max,\, deceleration} \cdot ||s_{j+1}||}). \tag{4.16}$$

Here we also need to take a minimum with $v_{j,f}$, because velocity cannot be more than the maximum allowed velocity in current curvature or maximum velocity in the current sample point when driving forward.

Now because we can calculate the velocity of the vehicle in each section start and endpoint we can calculate time to drive through the section. We assumed that acceleration is constant through the section so $a = \frac{v_j - v_i}{t_j - t_i}$, then

$$\begin{aligned}
||\mathbf{S}_j(t) - \mathbf{S}_i(t)|| &= v_i \Delta t + \frac{1}{2} a (\Delta t_{ij})^2 \\
||\mathbf{S}_j(t) - \mathbf{S}_i(t)|| &= v_i \Delta t + \frac{1}{2} \frac{v_j - v_i}{t_j - t_i} (\Delta t_{ij})^2 \\
2||(\mathbf{S}_j(t) - \mathbf{S}_i(t))|| &= 2 v_i \Delta t_{ij} + \frac{v_j - v_i}{\Delta t_{ij}} (\Delta t_{i,j})^2 \\
2||(\mathbf{S}_j(t) - \mathbf{S}_i(t))|| &= 2 v_i \Delta t_{ij} + (v_j - v_i) \Delta t \\
2||(\mathbf{S}_j(t) - \mathbf{S}_i(t))|| &= (2 v_i + v_j - v_i) \Delta t_{ij} \\
2||(\mathbf{S}_j(t) - \mathbf{S}_i(t))|| &= (v_i + v_j) \Delta t_{ij} \\
\Delta t_{ij} &= \frac{2 \cdot ||\mathbf{s}_{ij}||}{v_i + v_j}.
\end{aligned}$$

Now by summing times through all sections we get whole drive time

$$t = \sum_{j=1}^{n} \Delta t_{(j-1)j}. \tag{4.17}$$

It should be noted that n should be picked big enough because if we pick too small n, then whole curve can be between sample points and then it hasn't been taken account. In that case, the truck can drive too fast into that curve, because it hasn't be taken account. [12]

5 OPTIMIZATION ALGORITHM

In this chapter, we will take more close look into the optimization algorithm. For creating algorithm, we need to create fitness function, population initialization algorithm, crossover algorithm, mutation algorithm and define end criterias for the genetic algorithm. We will consider all the needed parts in different sections. We will also add example algorithms to each section. Algorithm is implemented by using the Python and that's why our example algorithms are also Python code. First, we will introduce the algorithm general functionality. After implementing the algorithm we ran test drives for the algorithm and document results.

5.1 Description

In this chapter, we will take a look into the optimization algorithm itself. We will take a closer look into population initialization, fitness function, and genetical methods. In the last subsection, we will also introduce the end criteria checking algorithm. First, we will take a look into population initialization.

5.1.1 Population initialization

In this subsection, we will go through the population's initialization. For the initialization, we use an starting spline. To ensure that the start point and endpoint of the spline remains the same we use the starting spline's first and last control point to every population candidate. We also need to ensure that angle at the start and endpoint remains the same. Because of that we copy also second control point and second last control point of the starting spline to every population candidate and we only modify inner control points $(2, 3, ..., n-4, n-3)$. This means that we need to generate $n-4$ control points for each spline. We need to assume that the spline contains at least 5 control points. The following figure describes the inner control points generation.

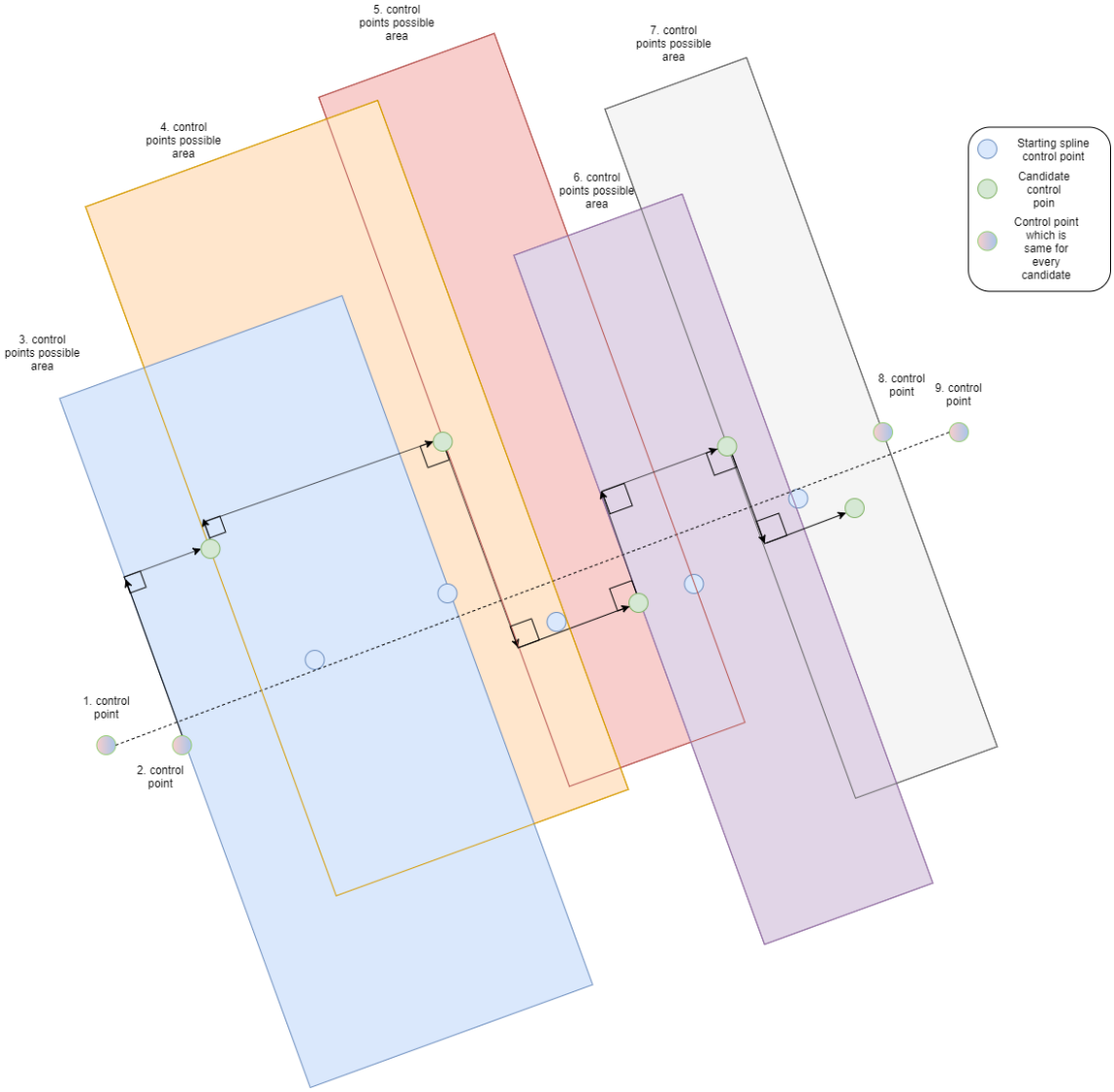


Figure 5.1. *Population initialization algorithm.*

In figure, two first and two last control points remains the same as they should and only inner control points are generated. In the figure, the rectangle describes the area where the control point can be generated. Algorithm for generating control points is described in appendix A.In population initialization algorithm we first calculate the slope between the first and the last control point which can be calculated as follows

$$k = \frac{y_2 - y_1}{x_2 - x_1}. \tag{5.1}$$

We know that a perpendicular slope is an opposite number of the inverse of the slope. A new generated control point coordinate can be calculated in two steps. The first by moving along the perpendicular line and after that moving the parallel line. At the first step we need to calculate the new point in the perpendicular line. This point is obtained by moving a random distance along the perpendicular line from the previously created control point. If we are creating the first control point, we start from the 2nd control point of the original spline. Next we have to define the maximum distance for both directions, because there

is no point creating control points with the non restricted distances. By studying the behaviour of the splines when control points is moved gives good direction to the maximum distance. Also in many cases optimal solution is quite near the starting spline and in tight warehouses obstacles and other routes will be near the spline. Because of these facts we select the maximum distance to be quarter of the distance from the first control point to the last control point. By defining the maximum distance in this way, population generation is not dependant on inserting any limitations. We calculate point x-coordinate same way as in equation (4.7). Because that gives us two x-coordinates (both directions of the line), we need to randomly select one of the directions. y-coordinate can be calculated by inserting previously calculated x-coordinate to line formula of the perpendicular line.

The second step is to move from previously calculated point along the parallel line to the line between the first and the last control points. The first we need to randomly select distance we are going to move along the line. Maximum distance is the distance from lastly created control point to the $i+1$:th control point of the original spline. Distance is not straight distance from point to the point but it is the distance of the projection to the line from the first control point to the last control point. That distance can be calculated by projecting previously created point to the line which is parallel to the line from the first control point to the last control point and goes through the $i+1$th control point of the original spline. Now let (x_1, y_1) be the previously calculated point and let $\mathbf{p_{i+1}}$ be the control point. Now line formulas are

$$y = \frac{p_{n,y} - p_{0,y}}{p_{n,x} - p_{0,x}}(x - p_{i+1,x}) + p_{i+1,y} \tag{5.2}$$

and

$$y = \frac{-1 \cdot (p_{n,x} - p_{0,x})}{p_{n,y} - p_{0,y}}(x - x_1) + y_1. \tag{5.3}$$

Let's mark $k = \frac{p_{n,y} - p_{0,y}}{p_{n,x} - p_{0,x}}$. Now these lines cross point x-coordinate is the point where y-coordinates are same, so

$$\begin{aligned}
k(x - p_{i+1,x}) + p_{i+1,y} &= -k^{-1}(x - x_1) + y_1 \\
kx - kp_{i+1,x} + p_{i+1,y} &= -k^{-1}x + k^{-1}x_1 + y_1 \\
kx + k^{-1}x &= kp_{i+1,x} - p_{i+1,y} + k^{-1}x_1 + y_1 \\
x(k + k^{-1}) &= kp_{i+1,x} - p_{i+1,y} + k^{-1}x_1 + y_1 \\
x &= \frac{kp_{i+1,x} - p_{i+1,y} + k^{-1}x_1 + y_1}{k + k^{-1}}.
\end{aligned}$$

y-coordinate can be calculated by inserting this x-coordinate into either line formula. Now the maximum distance is

$$w_{max} = \sqrt{(p_{i+1,x} - x_2)^2 + (p_{i+1,y} - y_2)^2}. \tag{5.4}$$

Next, we will use the same formula as previously for moving along the line, but now our slope and starting point is different. The slope is the slope of the line between the first and

the last control point and the starting point is the previously calculated point (x_1, y_1). Now equation gives us two x-coordinates as previously, but this time we can't pick a random direction, because we want control points to move closer to the last control point. So we need to pick the direction where new x-value is closer to $i+1$th control point.

5.1.2 Fitness function

For the fitness function, we want to take account drive time and all other restrictions of the path. Cases where path violates some restrictions we want to penalize them much so that their fitness value is very high. Because of that, we create a so-called "penalty" function to be an exponential function. For the penalty function, we can select

$$p(c) = e^{20\cdot(c-0.9)}. \tag{5.5}$$

When $0 \leq c \leq 1$ function $p(x)$ returns small values and when $c > 1$ function returns big values, because function is exponentially growing function. [12] This fits very well to the fitness function because we need the way to compare different paths for each other and this function makes illegal paths to have very big fitness value, which makes them the worst candidates. For the checking, if the path doesn't cross not allowed area we use the custom function. For the path validity checking we use the function:

$$g_j = \begin{cases} p(10) & \text{, if collision happened} \\ 0 & \text{, else.} \end{cases} \tag{5.6}$$

If not allowed area border intersection point is in line from the sample point to safe area border, then it means that path is not valid and we need to penalize the path. We also want to check the wheel turning angle restriction in the fitness function. We set a penalty to be high because the path should be valid. This is done by compairing wheel current steering angle at every curve sample point to maximum allowed turning angle. Because wheel can turn in both directions, we need to take norm from the current angle. We will use same penalization function as for collision check but now we insert the norm of current steering angle divided by the maximum steering angle. Now if steering angle exceeds the maximum allowed value, then $\frac{||\phi_j||}{\phi_{max}} > 1$ and then penalty function grows much. So now the whole fitness function in general form is

$$f = t_{drive} + \sum_{j=1}^{n} (p(\frac{||\phi_j||}{\phi_{max}}) + g_j), \tag{5.7}$$

where t_{drive} stands for a drive time of the path, ϕ_j is steering angle in the current sample point and ϕ_{max} is maximum steering angle. [12] Fitness value calculation algorithm is introduced in appendix D. For the calculating fitness value, all you need to define is length of the vehicle, sample point amount n, maximum steering angle and minimum safe distance to not allowed area. More restriction checkings can be easily added into fitness

function. All you need to do is to add restriction checking in to sum function same way than wheel turning angle checking is added.

5.1.3 Genetic operations

Genetic operations are key to a successful genetic algorithm. Before developing a genetic algorithm, we need a parent selection algorithm. Parent selection algorithm gives two random parents from the population which is used for creating a new candidate into the new population. The following algorithm describes the parent selection algorithm.

```
1  def select_random_parents(betterParents,lowParents):
2      parents = []
3      for i in range(2):
4          parent = object()
5          if random.uniform(0,1) < (1 - favor_better_parents_P):
6              max_parent_ind = len(bad_parents)-1
7              parent_index = random.randint(0,max_parent_ind)
8              parent = bad_parents[parent_index]
9              bad_parents.remove(parent)
10         else:
11             max_parent_ind = len(good_parents)-1
12             parent_index = random.randint(0,max_parent_ind)
13             parent = good_parents[parent_index]
14             good_parents.remove(parent)
15         parents.append(parent)
16     return parents
```

Listing 5.1. *Parent selecting functionality.*

The first, algorithm selects parent 1 from the population such that the algorithm weights better parents by P_{favour} parameter. This means that it is more likely to pick a "better" parent than "worse" parent. After parent 1 is selected, parent 2 is selected the same way but the only difference is that now parent 2 can't be the same as parent 1. It is also possible to select more parents in the same way as parent 2 is selected. From nature comes that we pick 2 parents for the new candidate construction, but researches have been shown that in some cases selecting more parents would be much more effective. Now when we have selected parents we will go through the crossover algorithm and the mutation algorithm.

Crossover

Crossover method is one of the genetical operations and it is used for creating the new candidates into the new population from the current population. The crossover operation

for this algorithm is described in the following figure.

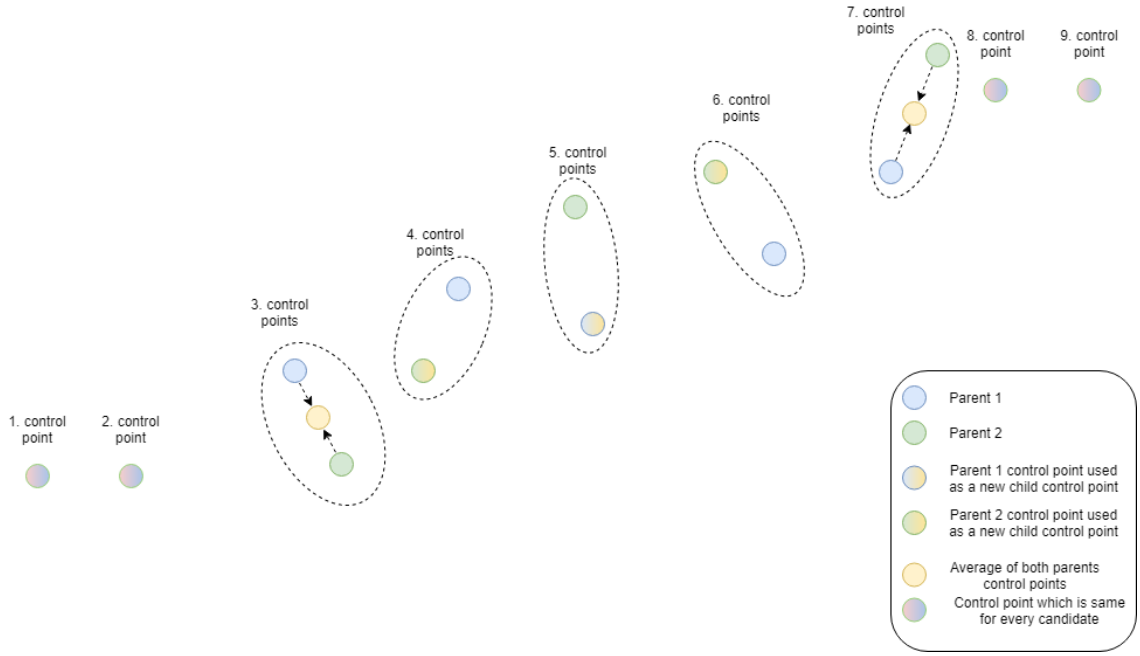


Figure 5.2. *Crossover algorithm.*

For all candidates, the first and the last two control points are the same. So all we need to do is to create middle control points. For the crossover operation, we need to first define new candidates' parents. For this system, we select two parents, but this also works for more parents. After we have the parents selected, then for each parent we number control points from 1 to nth control point. Then after that, we create new candidates' control points. New candidates' ith control point is always one of the parents ith control point or the average of the parents ith control points. We don't weight any possibility, we just randomly pick one of them for each control point. The algorithm is introduced in appendix B. It is really easy to add more parents to this method. The next parents are added the same way as parent 2 is in the operation.

Mutation

The mutation is the second genetic operation. Mutation describes nature's mutation functionality. Mutation operation is used for randomly picked new candidates in the new population. A number of mutated candidates depend on the mutation probability variable. Mutation creates variety into the population and sometimes mutation changes populations direction to the better. The following figure illustrates the mutation process in this algorithm.

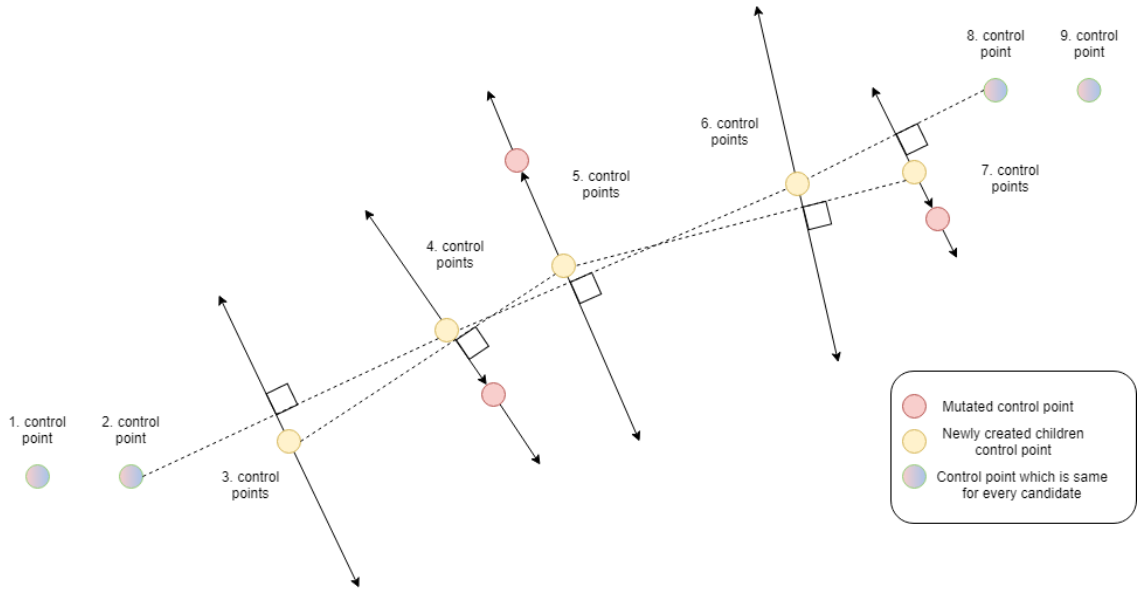


Figure 5.3. *Children mutation algorithm.*

At the first in the mutation operation, we randomly select the number of mutated control points. After that, we randomly select the control point which will be mutated. For the mutating the control point, we need to first calculate the slope of the line between previous and the next control point and also the distance between those control points. After those are calculated, we mutate the selected control point by moving it along the line which is perpendicular to the line between previous and the next control point. The moved distance is randomly picked between 0 and half of the previously calculated distance. The moving direction is also randomly picked. The algorithm is described in appendix C.

New point x-coordinates are calculated by using the formula (4.7) and y-coordinate is calculated by inserting that x-coordinate to the line formula. For the x-coordinate we have that

$$x_2 = \pm \frac{d}{\sqrt{1 + (-1 \cdot \frac{p_{i+1,x} - p_{i-1,x}}{p_{i+1,y} - p_{i-1,y}})^2}}. \tag{5.8}$$

Respectively for the y-coordinate we have

$$y_2 = -1 \cdot \frac{p_{i+1,x} - p_{i-1,x}}{p_{i+1,y} - p_{i-1,y}}(x_2 - p_{i,x}) + p_{i,y}. \tag{5.9}$$

This new coordinate is calculated the same way for all mutated control points. The maximum number of mutated control points is always $n - 4$, because the first two and the last two control points are always the same for all candidates.

5.1.4 End criteria

This section contains a clear description of the end criterion. Clear and realistic end criteria ensure that the algorithm will end and it will end when it has to stop. Next table

introduces all the possible ending cases.

End criteria	Description
n_{max}	Maximum number of rounds
$n_{f_{max}}$	Best fitness value remains same too many rounds
$P_{f_{imp}}$	Best value improvement is big enough

Table 5.1. *End criteria parameter descriptions.*

Basically, there are three cases that have to be checked. First is the maximum number of rounds. This criterion ensures that the algorithm will eventually stop, even the best fitness value is not good enough. This criterion prevents the infinite loop. The second criterion prevents the best fitness value staying the same too many rounds. This ends the algorithm if the local maximum is found and it cannot grow anymore. Setting this value too low makes the algorithm stop too early and setting this value too big make an algorithm to run around the same point long. The third criteria is criteria for making possible to improve route only little bit much faster, because we don't have to run whole optimization process in to end. Improvement is calculated by comparing the current best fitness value to first spline fitness value. Setting improvement low makes algorithm create small improvements fast and if you want to find the best possible value you have to set this improvement value big enough so that this end criteria is not met. Next program illustrates how end criteria are checked in the implemented program.

```
1  def IsEndCriteriaMet(currentRound,sameBestValueRound,
2                       currentBest,startFitnessValue):
3
4      # Calculate fitness value improvement
5      valueImprovement = currentBest / startFitnessValue
6
7      if currentRound >= n_max or
8         sameBestValueRound >= n_f_max or
9         valueImprovement >= P_f_imp:
10         return true
11     return false
```

Listing 5.2. *End criterion checking.*

All the end criteria variables are important to adjust to correct magnitude because if one of those values is too small it can make the algorithm stop too early and optimal value is never reached.

5.1.5 Current implementation solutions

In implementing the algorithm we noticed a few remarks which are handled in this subsection. Those remarks effect mostly into algorithm efficiency. In the algorithm, some candidates survive to the next population based on their fitness value. In population evaluation it should be checked that we only evaluate and calculate fitness values for the new candidates. Because if we evaluate all the candidates in all populations, then number of candidates pointlessly evaluated is $N_b(r-1)$, where N_b is the population base size and r number of rounds. This increases population evaluation time and also increases the execution time of the algorithm.

Evaluation of the population could be implemented concurrently. Concurrent evaluation can decrease the execution time of the algorithm, but the benefit of the concurrent evaluation depends only on the running system specifications. Population initialization could be also done concurrently, but in this case, we need to calculate before how many new candidates every thread creates. Also, collision detection is possible to implement concurrently. In collision detection, every thread checks only one group of the obstacles. In this implementation, we only implemented the population evaluation concurrently and tested that if the concurrency decreases execution time. Results can be found in the next section.

5.2 Results

This section contains algorithm testing results. We will concentrate on a few important properties of the algorithm more closely and after that, we will test the whole algorithm.

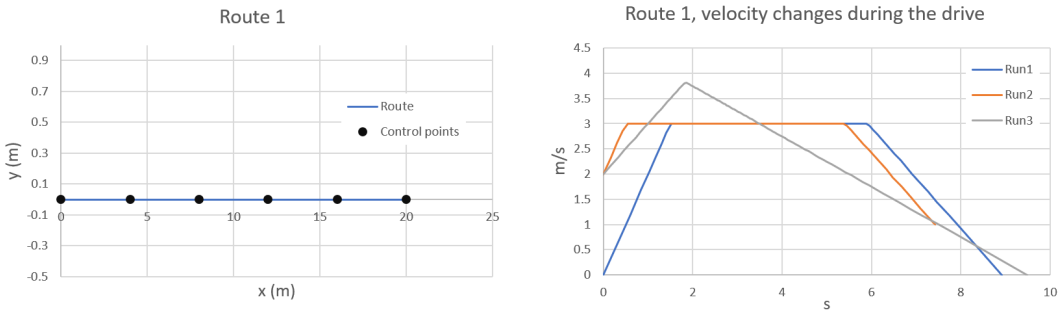
5.2.1 AGV velocity evaluation

In this section, we are going to concentrate on testing the AGV's velocity calculation during the route. In this test, we test that AGV's velocity doesn't exceed maximum velocity, we also check that the AGV's acceleration doesn't get exceeded and deceleration doesn't exceed maximum allowed decelation. We will test velocity evaluation with five different types of routes. The first one will be very simple straight line, which is parallel to x-axis, the second route will be also simple straight line, but it will be parallel to y-axis, the third route will contain few big curves, the fourth route will contain many curves with different sized curves and the last one will be the same kind of than the fourth, but it will go to different direction. We will test all routes three times with different types of settings. Those settings are listed in the following table.

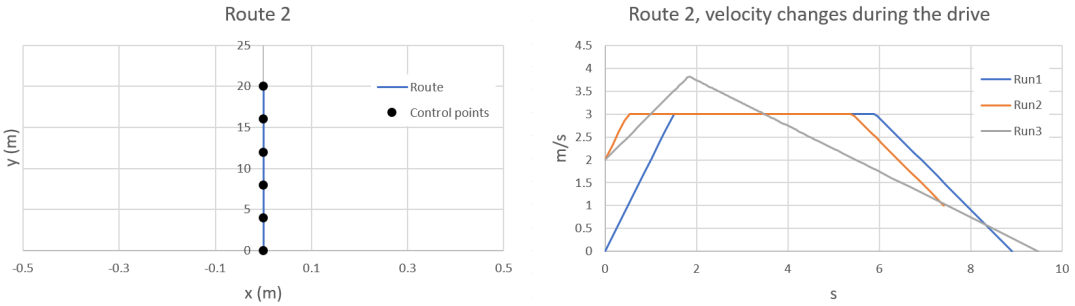
	Run 1	Run2	Run 3
Maximum velocity (m/s)	3.0	3.0	7.0
Maximum acceleration (m/s^2)	2.0	2.0	1.0
Maximum deceleration (m/s^2)	1.0	1.0	0.5
Starting velocity (m/s)	0.0	2.0	2.0
End velocity (m/s)	0.0	1.0	0.0

Table 5.2. *The AGV settings during the velocity test runs.*

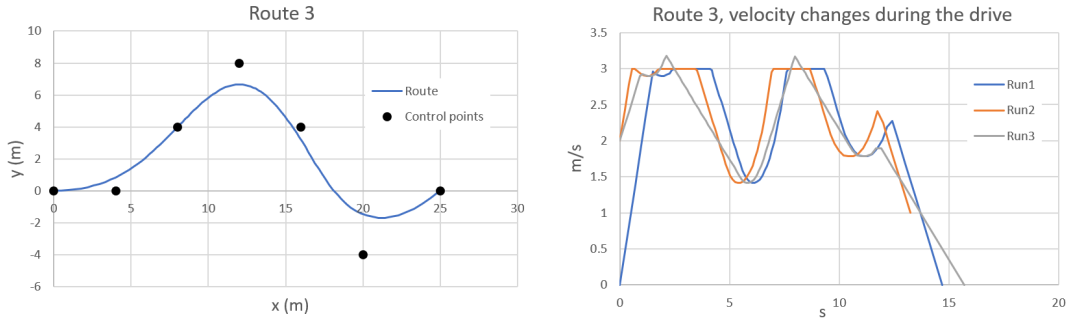
Results from the tests are given in the following figure. The figure contains an illustration of the route and it also contains a diagram of the velocity changes respect to the drive time.



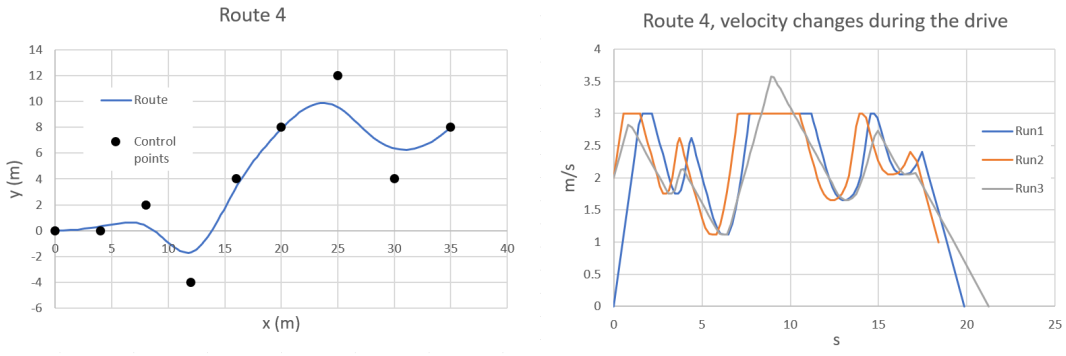
***(a)** Simple straight route, where route goes parallel to x-axis.*



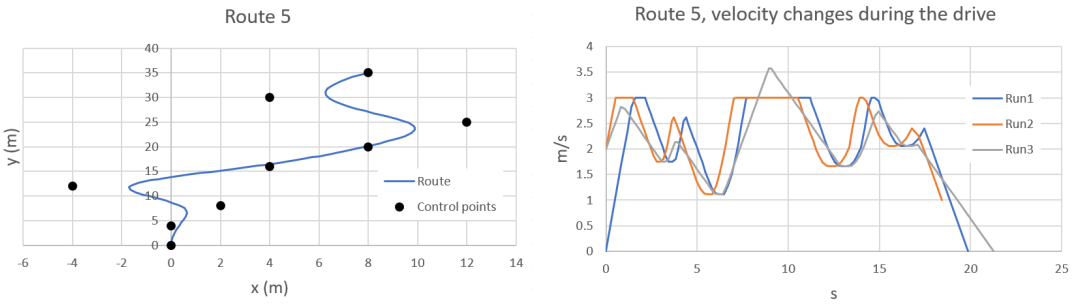
***(b)** Simple straight route, where route goes parallel to y-axis.*



***(c)** Route which contains few curves.*



***(d)** Complex route which contains multiple different sized curves.*



***(e)** Same complex route but now it goes to different direction.*

***Figure 5.4.** AGV velocity evaluation tests results diagrams.*

As we can see from the above diagrams different AGV settings affect the shape of the velocity diagram as they should. The first two routes are straight lines so the velocity diagram should be quite simple. The first AGV accelerates into top velocity if it is possible to reach in given distance and after that AGV will start to decelerate to reach the end speed. In both cases, drive distance is the same 20m, so velocity diagrams should be the same. As we can see from the above diagram, they look the same. Velocity diagrams also look correct because in the first run AGV starts from 0 m/s and stops into 0 m/s and respectively in the run 2 start velocity is higher and end velocity is also higher, so in run 2 AGV reach the end of route much faster than in run 1. In run 3 we decreased maximum acceleration and deceleration, but we increased maximum velocity. As we can see from the figure the AGV will reach higher velocity, but it has to start deceleration much earlier and it is much slower so it takes much more time to AGV reach the end of the route. After looking at the first two route diagrams it seems that velocity diagrams are correct for both routes. In more complex cases, we can see clear drops in the velocity at the curves. In all cases, velocity seems to behave correctly and results are expected.

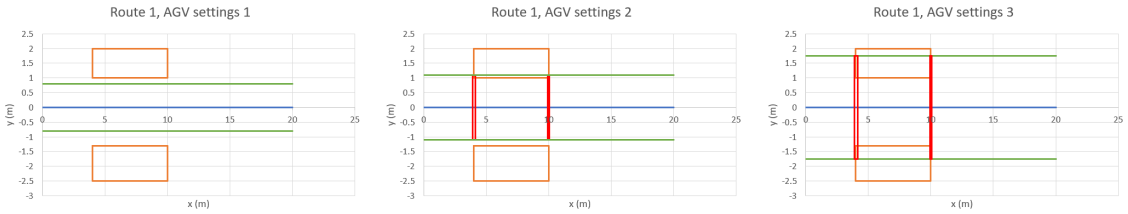
5.2.2 Vehicle collision detection

In this test, we will test the collision detection algorithm. Collision detection will be tested in four different routes. For the testing, we will use routes 1, 2, 3 and 5 from velocity tests. In this test, we will change AGV width setting and AGV safe area width setting. All other settings don't affect this test, because we have defined safe area to be depent only on those properties. As previously said that is not the case in real-world, because in real-world also AGV length would affect in this test, but we will talk more about this problem later. Tests settings are listed in the following table.

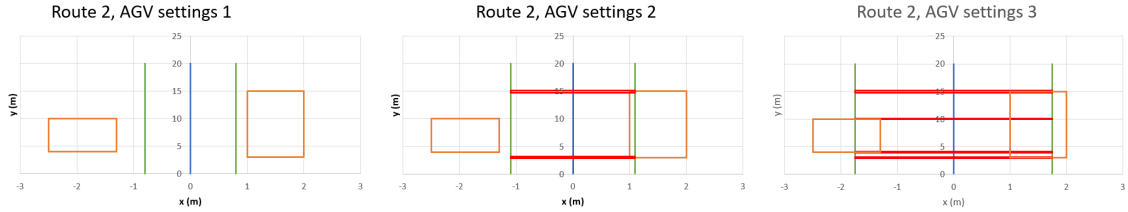
	Run 1	Run2	Run 3
AGV width (m)	1.0	1.5	1.0
AGV length (m)	3.3	3.3	3.3
Safe area width (m)	0.3	1.0	0.6

Table 5.3. *The AGV settings during the collision detection test runs.*

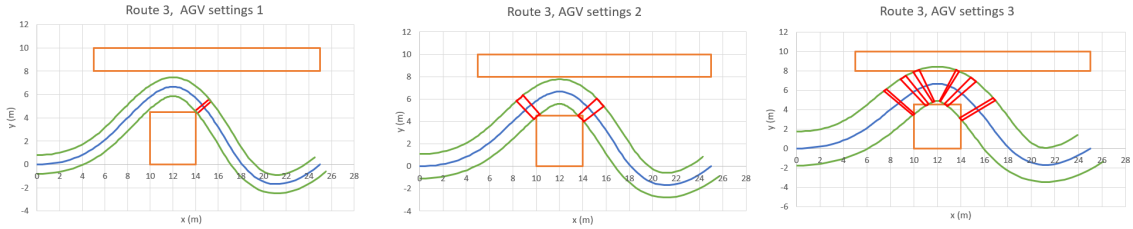
Results from the tests are given in the following figure. The figure contains an illustration of the route, not allowed areas, safe area boundaries and collision points.



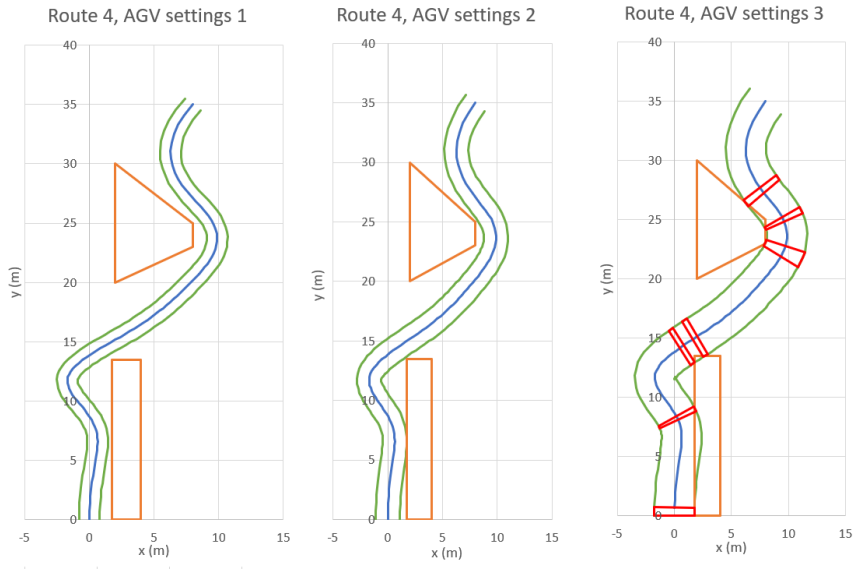
(a) *Simple straight route, which goes parallel to x-axis.*



(b) *Simple straight route, which goes parallel to y-axis.*



(c) *Route, which contains few curves.*



(d) *Complex route, which contains multiple different sized curves.*

Figure 5.5. *Collision detection test result figures. Blue line is route, green lines are bounds of safe area, orange lines are not allowed areas boundaries and red boxes indicates sections where collision is detected.*

As we can see from previous figures, collision detection detected at least one collision in the routes which contained at least one collision. Collision detection is made such that collision is detected if a safe area border intersects with obstacle border. This doesn't detect cases where a safe area moves inside the obstacle and cases where the whole

obstacle is inside of route. So there are few cases where collision might not be detected. Because of the nature of the algorithm, it is really easy to improve collision detection without needing to make changes to the algorithm itself. Because collision detection detects collision in most of the cases it is good enough for testing the algorithm itself. For the real system, it would be better to implement collision detection in some other method. One possible way of implementing collision detection is ray casting and it is shortly introduced in the next section.

5.2.3 Population initialization

Population initialization is an important part of the algorithm because it creates the basis for the algorithm. In population initialization, it is important to generate population which has variety into all directions. In this test, we test population generation for the 4 different routes. We will also test that population generation works the same way when running the algorithm multiple times. At first, we generate a population for a simple route, which is parallel to the y-axis. The next figure includes diagrams from that population's initialization.

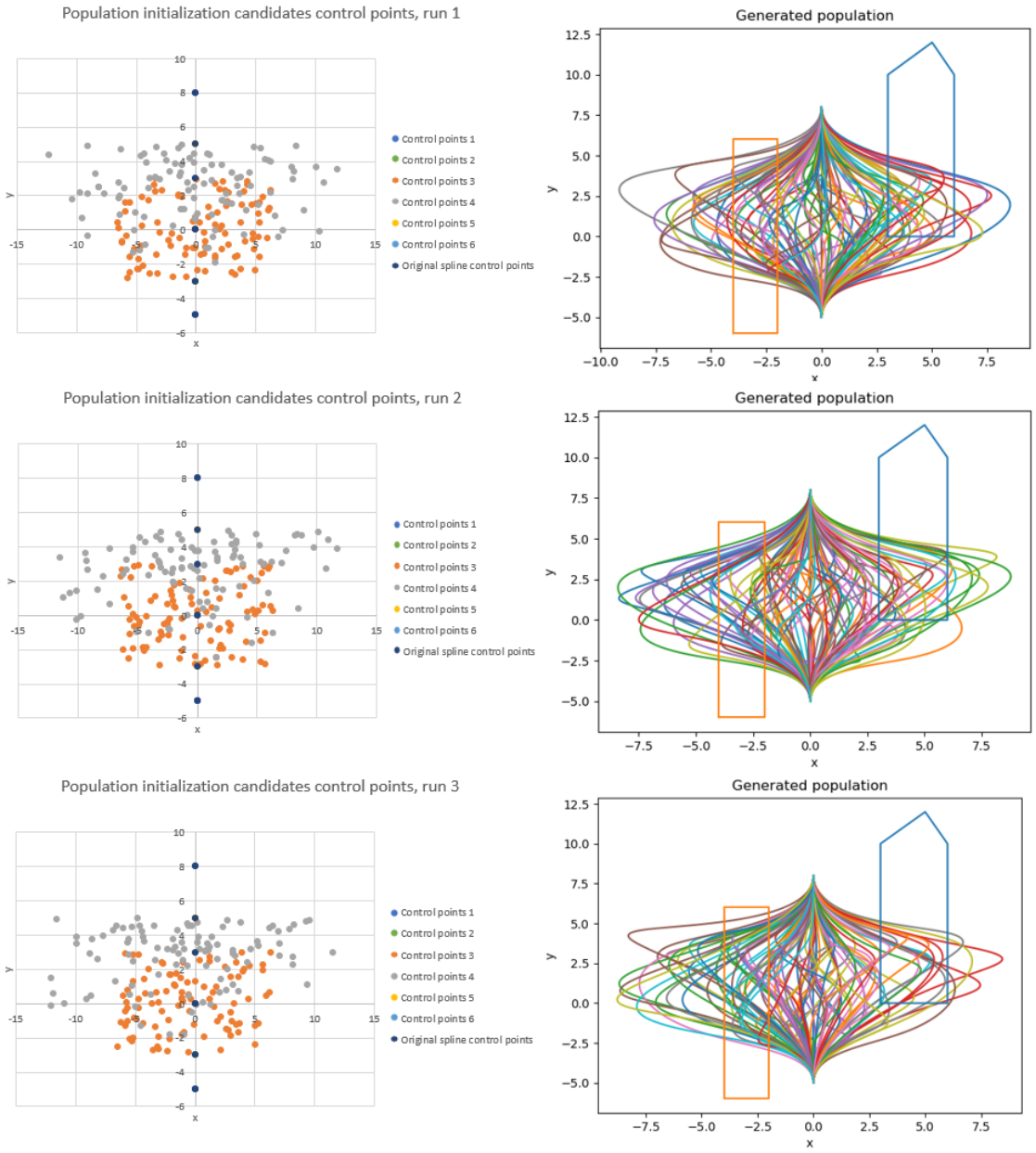
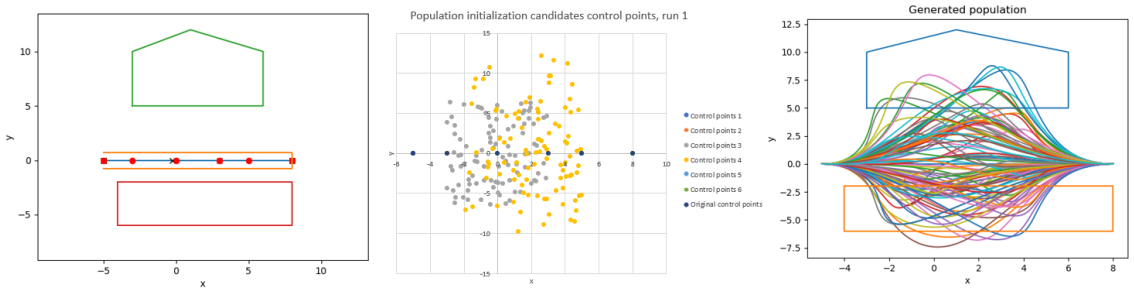
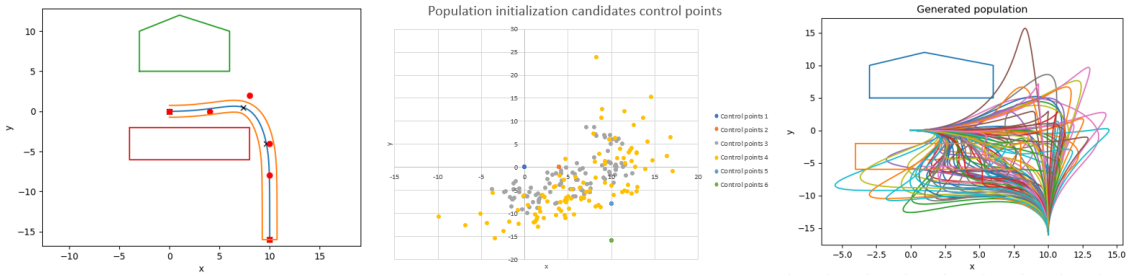


Figure 5.6. *Population generation for simple route which is parallel to y-axis. Population generation is ran multiple times to ensure that population generation works correctly when running it multiple times.*

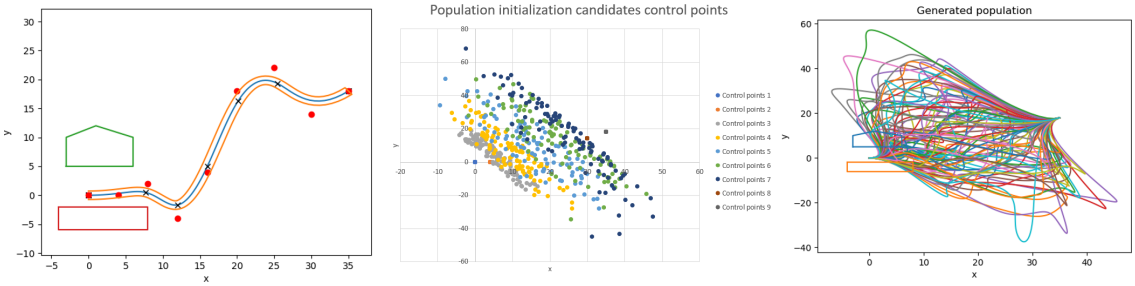
As we can see from previous diagrams population generation works the same way in all runs. In the previous case, only two middle control points are changed, because the first two and the last two control points are the same for all candidates. As we can see from the previous diagrams, only two middle control points are changed as they should. Control points are evenly distributed and because of that, population routes are evenly distributed. As we can see from the previous figures, all the routes start from the same point and end to the same point. Also, the start and end angle remain the same. Next, we will test the population generation for 3 different routes also. One of them is a simple straight route which is parallel to the x-axis and two of them are a little bit more complicated. The next figure contains diagrams from those population initialization tests.



(a) *Simple straight route, which is parallel to x-axis.*



(b) *Route, which contains one curve.*



(c) *Complex route, which contains multiple different sized curves.*

Figure 5.7. *Population generation for different routes. For each routes there is figure of route, diagram of all control points and figure of all generated routes.*

From the previous figure, we can see that in all routes population is evenly generated. When the number of control points is increased it seems that generated routes are more complex. In all cases, populations seem to be good enough for the algorithm and finding the optimal result. The next we will test population initialization times with different sized populations. Our goal is to find a relation between population size and the initialization time. We will run population initialization 30 times for each population size. We start from a 10 sized population and grow the population with 10 to the 90 sized population. The following figure shows the population initialization times for the different sized populations.

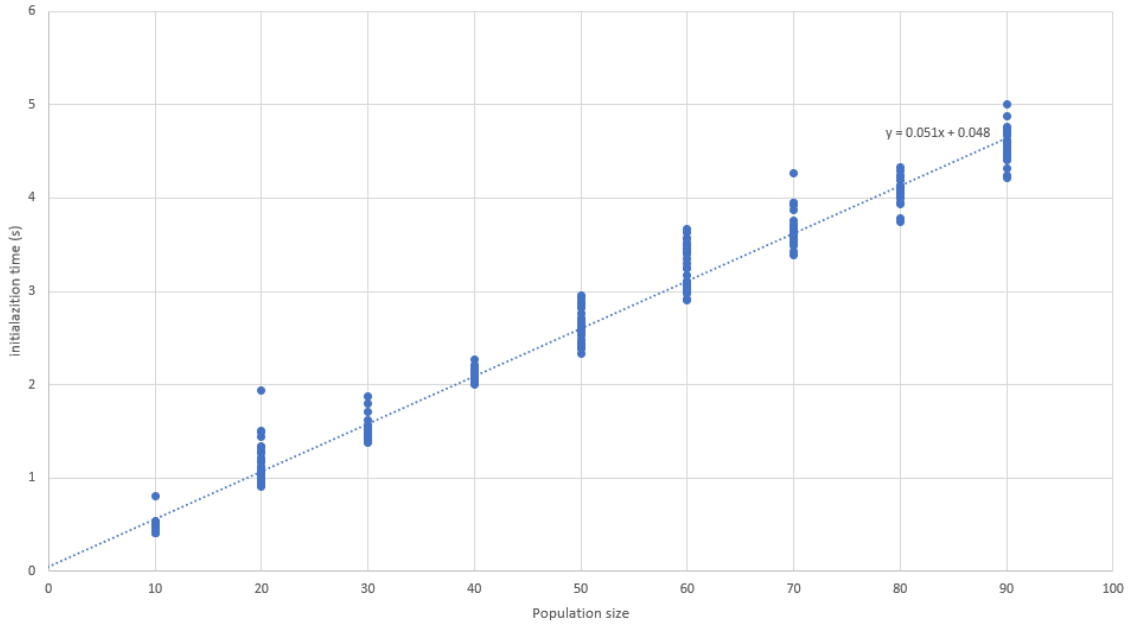


Figure 5.8. *Population initialization times with different sized populations.*

As we see from the figure, initialization time seems to be linearly growing when population size is growing. When fitting linear fit to the data, we get an approximation for the initialization time based on population size. For the initialization time, we have

$$t_{init} = 0.051N + 0.048, \tag{5.10}$$

where N is the population size. Population initialization time depends on the implementation of the population initialization, but in our implementation population initialization time seems to be linearly growing when population size is growing.

The next we will test population fitness value evaluation time with a different number of evaluation threads. Population evaluation could be done concurrently and next, we will try to find an optimal number of threads. The following figure contains evaluation times for different population sizes with a different number of evaluation threads.

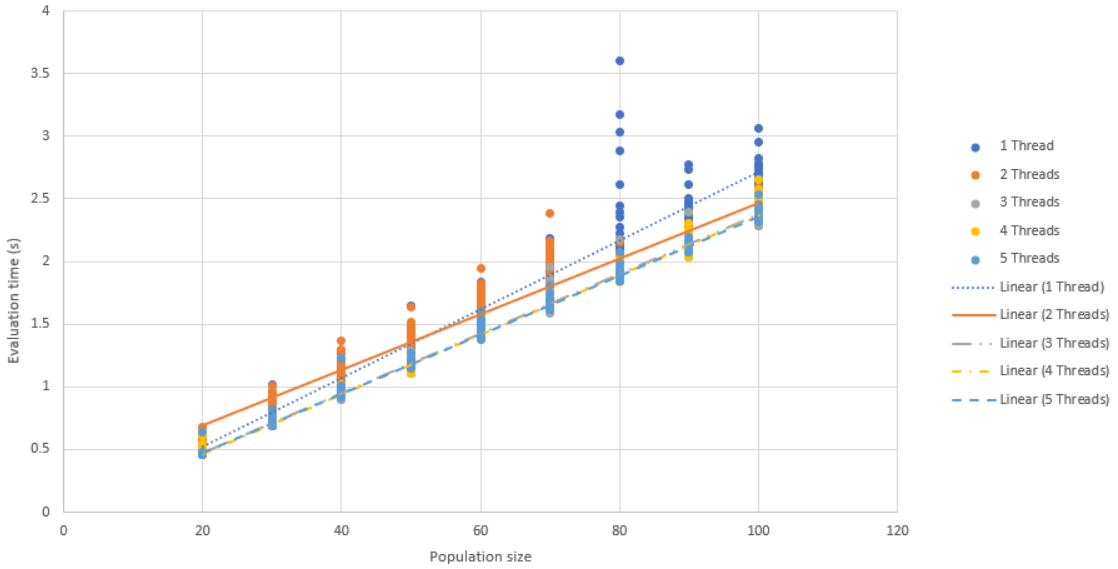


Figure 5.9. *Population evaluation times with different number of threads.*

From the test results, we can see that evaluation time seems to be linearly growing when population size is growing. Now by adding a linear fit to each dataset, we get approximations for the evaluation time for each population size with a different number of evaluation threads. As we can see from linear fits multiple threads give a small advantage when population size is big (>20). When dealing with small populations, using only one thread seems to be the good enough pick. There seems to be no difference after thread number grows over 3 threads. So the bigger populations it seems that 3 evaluation threads would be a good pick. However, we must remember that evaluation time and an optimal number of evaluation threads depend on the operating system.

5.2.4 Overall algorithm

Overall algorithm testing is made such that we try to optimize three different routes multiple times. The first case is the route where there are no obstacles and the optimal solution is just a straight line. The second route is the curve, where there are few obstacles and optimization doesn't require much changing the control points and the optimal route is a quite simple curve. The last one is a complex route that has multiple obstacles and the optimal route is not previously known. We ran every test multiple times to ensure that the algorithm works. We will also test algorithm working with different settings and try to find optimal population size and mutation probability for the algorithm. The settings used in tests are listed in the following table.

Maximum number of rounds	100
Maximum number of rounds the same fitness value	20
Population size	20,...,60
Parents ratio	0.5
Betters favour	0.7
Mutation probability	0.1,...,0.5
Number of spline parts	100

Table 5.4. *The Genetic algorithm settings during the tests test runs.*

As previously mentioned the first test case will not contain any obstacles. The starting route, in this case, will be the simple "bell curve". In this case, the optimal result will be the straight line between the starting and the ending point. The starting route's fitness value is $24,8327$. The following figure contains an illustration of the starting curve and the found optimal curve.

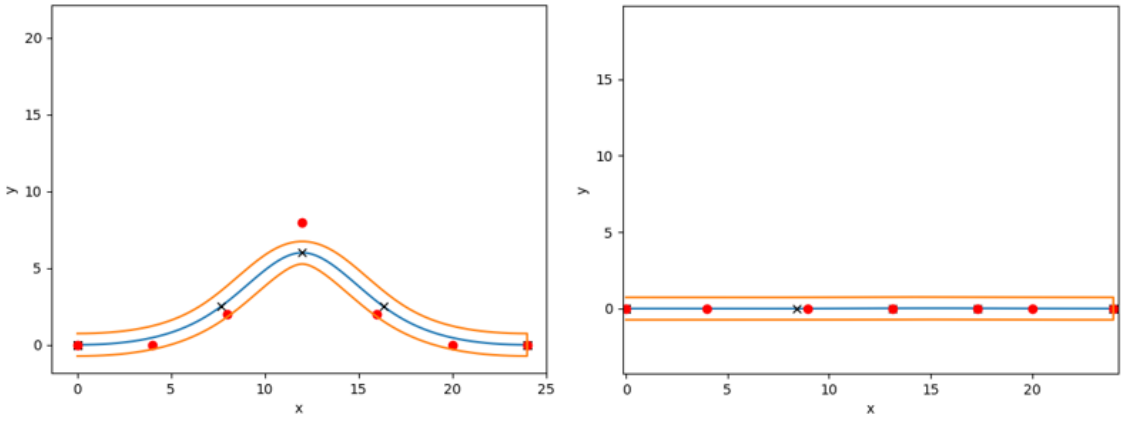


Figure 5.10. *The first test route of the overall genetic algorithm. The left route is starting route and the right one is the found optimal route.*

Optimization was run about 20 times with the same settings. The average optimization time was $91,38s$, an average number of optimization rounds were 89 round and the average fitness value improvement was $559,25\%$. Next, we test population size and mutation probability effect on the optimization algorithm. The following table shows results from those tests.

N	P_m	Avg exec. time (s)	Avg exec rounds	Fitness value $\pm$ STDEV
20	0.3	26.67	45	$4,4432 \pm 0,0085$
30	0.3	$37,86$	40	$4,4401 \pm 0,00003$
40	0.3	$56,31$	42	$4,4399 \pm 4,69 \cdot 10^{-5}$
50	0.3	$66,69$	40	$4,4399 \pm 0.0001$
60	0.3	$123,00$	39	$4,4399 \pm 0,0001$
30	0.1	$44,03$	45	$4,4400 \pm 0,0002$
30	0.2	$41,01$	39	$4,4403 \pm 0,0008$
30	0.3	$51,12$	47	$4,4401 \pm 0,0004$
30	0.4	$50,12$	45	$4,4402 \pm 0,0004$
30	0.5	$41,37$	37	$4,4401 \pm 0,0004$

Table 5.5. *Route 1 test results.*

As we can see population size doesn't have a big effect on the found result. Population size seems to have a big impact on average evaluation time, but also to the standard deviation of the fitness values. In most cases, an average number of rounds is close around the 50 rounds. There can saw a slight impact in population size to the standard deviation of the fitness values. In small populations, there is a slightly bigger dispersion in fitness values. Mutation probability has a bigger effect on the found result and average execution time. It seems that small mutation probability grows the dispression in the found optimal solutions. This means that the optimal solution is not found as often. When the standard deviation is small, the same optimal solution is found more often.

The next we will test population fitness values evolution during the optimization. The following figure contains a diagram of all fitness values during the optimization.

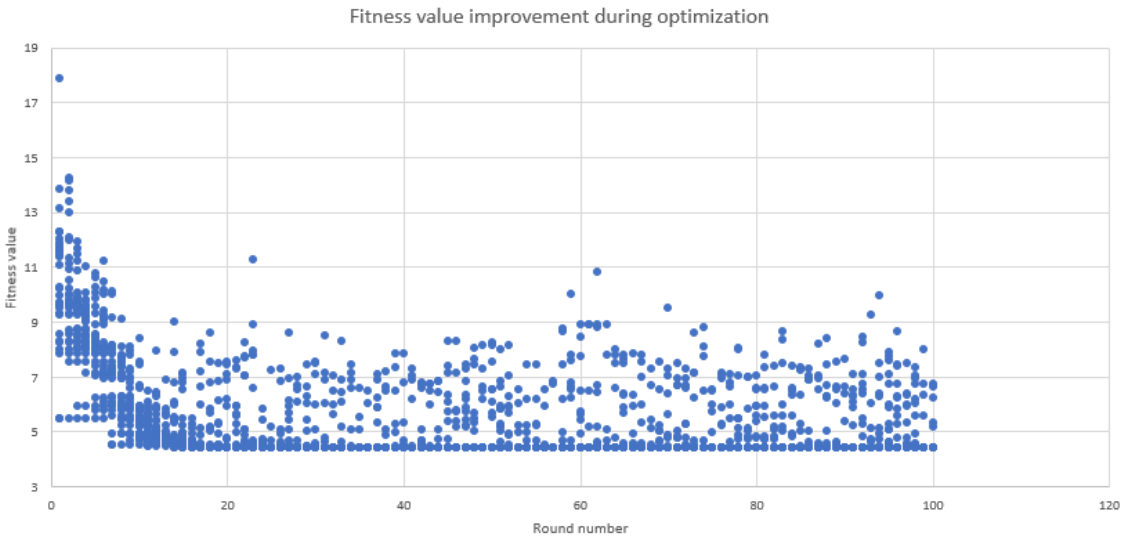


Figure 5.11. *Population fitness values evolution during the optimization process.*

As we can see from the diagram, after about 15 rounds there is no remarkable improvement in the population's best fitness value. There is a slight improvement that prevents fulfilling any end criteria. It seems that it would be good to check also that if the improvement is under 0.1%, counters are not reset. Let's add that to the algorithm and run the same test again. The following figure has the same results of the new test.

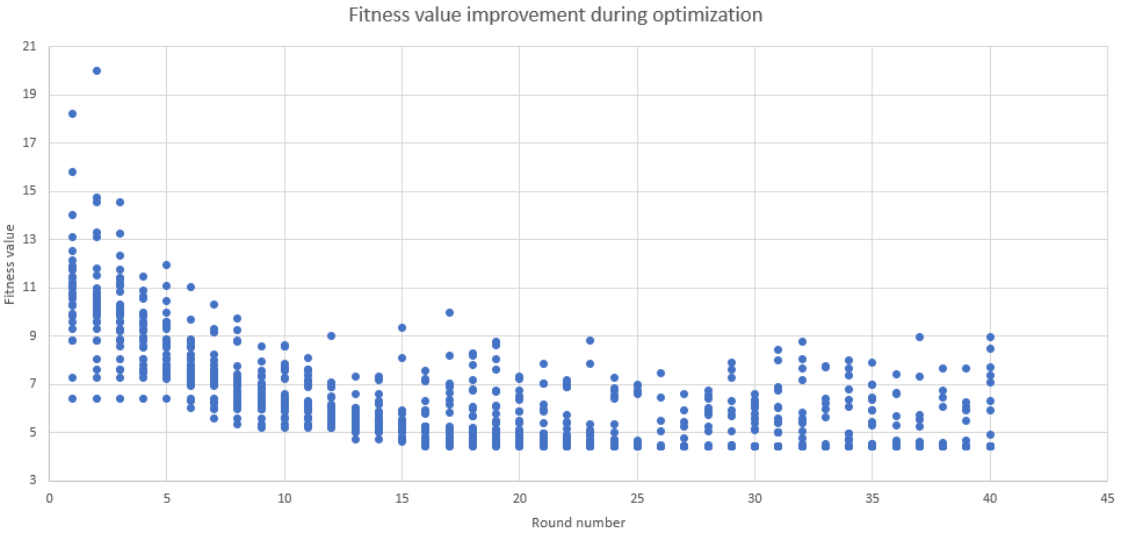


Figure 5.12. *Population fitness values evolution during the optimization process, when* 0.1% *improvement doesn't reset counter.*

As seen from the diagram now algorithm stops much faster as wanted. Now the average optimization time was $32.299s$, an average number of rounds was 41 and the average improvement was $559,29\%$. It seems that in this case, we reach as good average improvement with lower average time and lower round average as the previous test. This modification might affect more situations where the algorithm stops too early and the optimal result is not found yet. Let's test genetic algorithm functionality with two more routes

which both contains also obstacles where AGV can collide.

Now for the second route, we will run the same tests as for the previous route with the same settings. Now we will straight use the new improvement minimum condition, which we added in the previous test. The following figure contains the illustration of the start route and one of the found optimal routes.

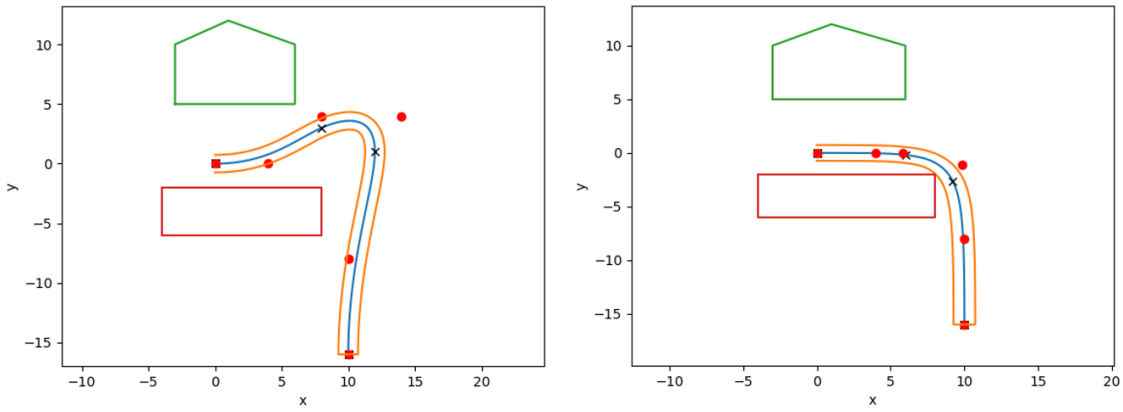


Figure 5.13. *The second test route of the overall genetic algorithm. The left route is the starting route and the right one is the found optimal route.*

We will test the algorithm with 5 different population sizes and five different mutation probabilities the same way as in the previous case. The starting spline's fitness value is $20,9948$. The following table has the results of the tests.

N	P_m	Avg exec. time (s)	Avg exec rounds	Fitness value $\pm$ STDEV
20	0.3	$54,30$	66	$6,4330 \pm 0,1012$
30	0.3	$89,56$	66	$6,3857 \pm 0,0599$
40	0.3	$117,58$	65	$6,3820 \pm 0,0533$
50	0.3	$216,33$	66	$6,3880 \pm 0,0602$
60	0.3	$207,74$	79	$6,3462 \pm 0,0158$
30	0.1	$83,01$	63	$6,4475 \pm 0,0998$
30	0.2	$130,90$	73	$6,4176 \pm 0,1036$
30	0.3	$123,57$	69	$6,4400 \pm 0,1184$
30	0.4	$127,21$	69	$6,3678 \pm 0,0377$
30	0.5	$159,03$	59	$6,4083 \pm 0,0884$

Table 5.6. *Route 2 test results.*

As we can see from the test results growing the population size decreased fitness value and the standard deviation of the fitness value. Also, population size increasing, increased the average of execution rounds and average execution time. In mutation prob-

abilities, there can't be seen a clear effect in fitness value or standard deviation of fitness values, when mutation probability is changed. For this optimization problem, it seems that the best mutation probability value would be between 0.4 and 0.5.

For the third route, we will run the same tests as for the previous routes with the same settings. The next figure contains the illustration of the starting and found the optimal route.

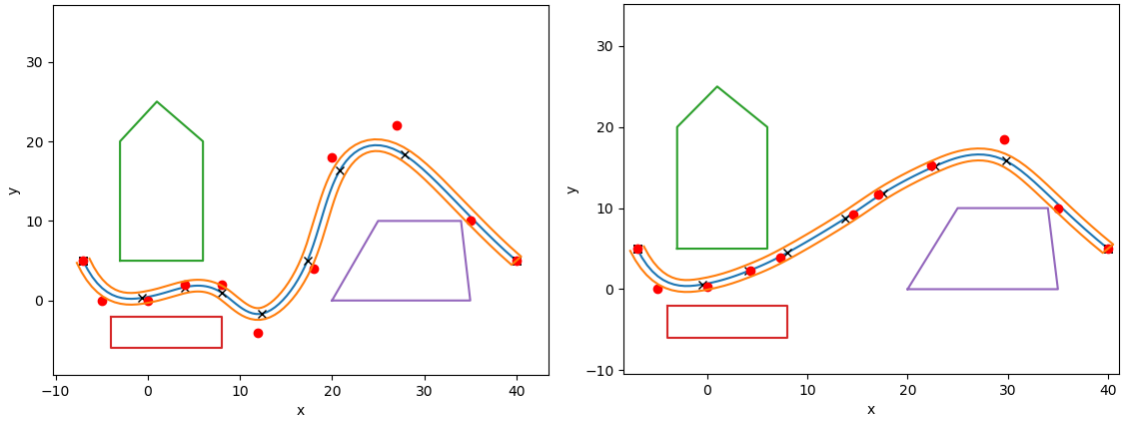


Figure 5.14. *The third test route of the overall genetic algorithm. The left route is the starting route and the right one is the found optimal route.*

As we can see from the previous figure route 3 contains multiple curves and obstacles. Starting splines' fitness value is $22,4642$. The following table contains the test results of the route 3.

N	P_m	Avg exec. time (s)	Avg exec rounds	Fitness value $\pm$ STDEV
20	0.3	$97,95$	100	$15,5980 \pm 0,8076$
30	0.3	$126,32$	92	$15,7172 \pm 2,4533$
40	0.3	$284,08$	100	$15,0356 \pm 0,7154$
50	0.3	$293,23$	99	$14,7616 \pm 0,6502$
60	0.3	$394,57$	100	$14,5322 \pm 0,6048$
50	0.1	$246,77$	100	$14,6981 \pm 0,7578$
50	0.2	$311,73$	97	$14,5475 \pm 0,3619$
50	0.3	$244,13$	100	$14,6335 \pm 0,5570$
50	0.4	$328,89$	99	$14,9285 \pm 0,5094$
50	0.5	$268,50$	100	$14,7468 \pm 0,6224$

Table 5.7. *Route 3 test results.*

As we can see from test results, increasing the population size improves the best fitness value and decreases standard deviation of found best fitness values. Almost all cases,

average execution rounds is near 100, which is maximum number of rounds. This means that algorithm might end before it should and better fitness value could be achived with more rounds. It also can be seen from the test results that when average rounds increases near 100, it also increases average execution times compared to previous test runs. From mutation probability change tests, we can see that best fitness value average and standard deviation of fitness values decreases when mutation probability is changed from 0.1 to 0.2. Fitness value average and standard deviation of fitness values increases when mutation probability is changed from 0.3 to 0.5. It seems that best mutation probability for this problem would be around 0.1 and 0.3.

5.3 Development opportunities

This section contains the topics which will help to improve the current algorithm or give ideas for other possible usages for the algorithm.

5.3.1 Parent selection using fitness proportionate selection

Currently in the algorithm parent selection is made such that the population is divided into two different groups, better candidates and worse candidates. After that, we randomly select one of the groups, although such that it is more likely that we select better candidates group. After selecting the group, we select the parent randomly from that group such that every candidate in that group is as likely to be chosen. In this method, good parents can end up in a worse parent group when all the candidates' fitness values are close to each other. In this case, the parent should be picked equally distributed between all good parents.

For solving this problem we could use the fitness-proportionate selection, where candidates' probability to be chosen is based on their fitness value and they are not divided into two groups. This selection method is also called as a roulette wheel selection. Probability of the candidate to be chosen is

$$1 - \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}. \tag{5.11}$$

This method weights better candidates, but when population candidates' fitness values get closer, then those candidates' probability to be chosen is going to also closer and then they are going to be chosen as likely. [9] The following figure illustrates the candidate's probabilities to be chosen in both selection methods.

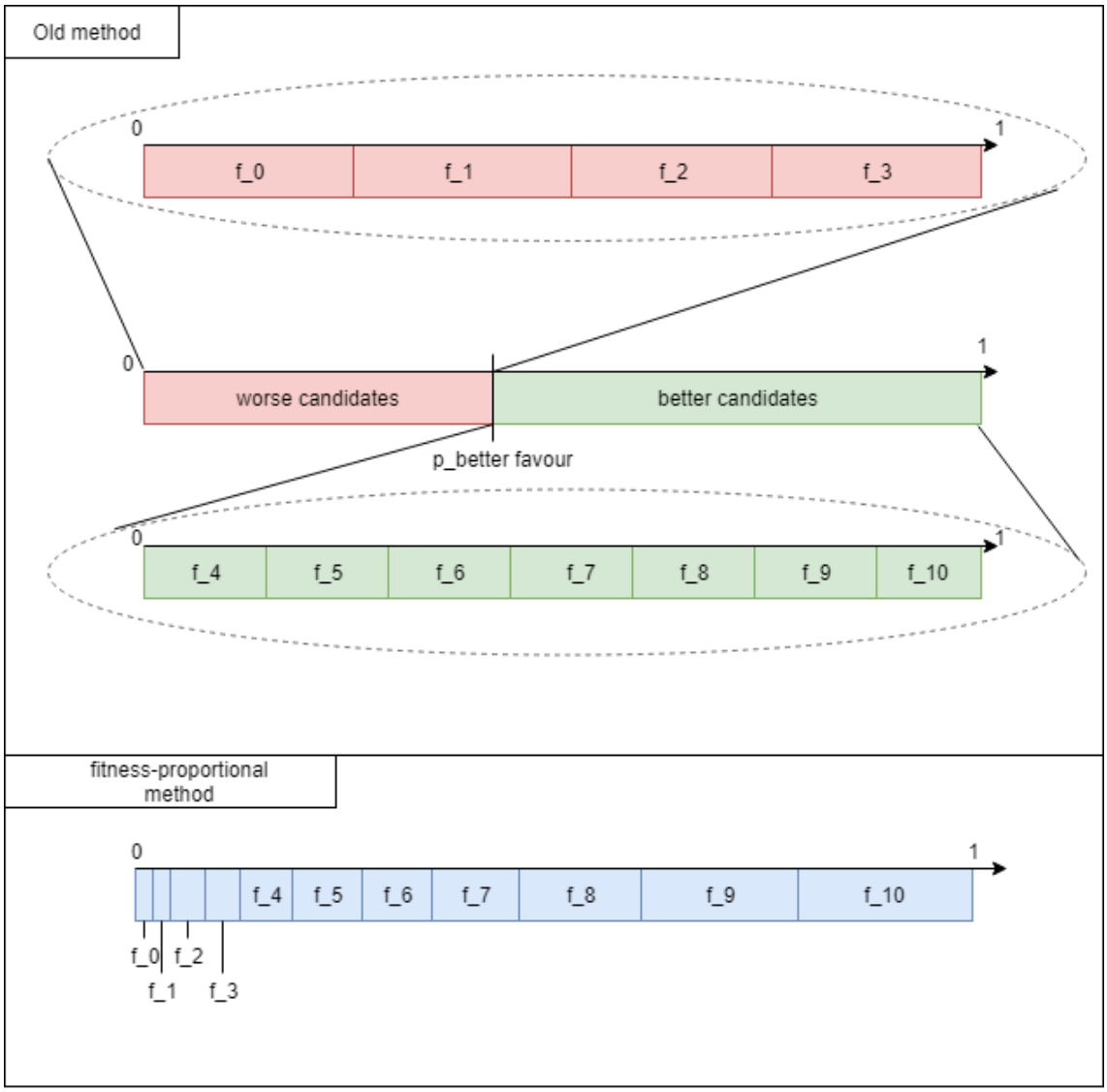


Figure 5.15. *Fitness-proportional selection method compared to implemented selection method.*

As we can see from the previous figure, all the candidates' selection probabilities are much closer to each other than in the fitness-proportional method. Fitness-proportional weights much better candidates than currently implemented method.

5.3.2 Collision detection and ray casting algorithm

As seen from tests, collision detection wasn't the best possible and it didn't notice cases when the route was going inside the not allowed area. For this optimization problem, it is enough, but for further development, it would be better to improve collision detection. If collision detection could be improved such that we can notice the amounts of cases when we are colliding to the not allowed area, then we would be able to penalize the fitness value amount of being collided. This will improve routes ordering such that routes, where collision times are smaller, are more suitable than cases where collisions happen

more often. Now in algorithm we only detect cases where route collides to the border and not cases where route goes inside the not allowed area. This affects that route which touch the not allowed area two times is as much penalized than route which drives long distances inside the obstacle. It follows that we won't be able to order routes by collision amount now very well.

One possible method for collision detection in this specific system is to handle routes' safe area and obstacles as polygons. Then we can check collisions by taking intersection with safe area polygon and obstacles. If the intersection is empty then the collision has not happened during the route. [16] We also tried this method at the first for this algorithm but at least for the Python language polygon intersection method took a long time and it wasn't used because of that.

Ray casting algorithm is a widely used method in game development to check object collisions. In this system and optimization algorithm, we are only interested in information about collision happening. We don't need information about the collision point, because collision determines only routes' validity. In game development, that information is very important, but because we don't need that information we won't focus on the theory of determining collision point. In the ray casting method, we use rays to determine if an object collides with another object. Ray is "shot" from object to different directions and then we calculate intersection times with different objects. If the ray intersects with the other object an odd number of times, then the object is inside the colliding object and respectively if the ray intersects even-numbered times the object, then the object is the outside of the colliding object. The following figure illustrates the ray casting algorithm functionality.

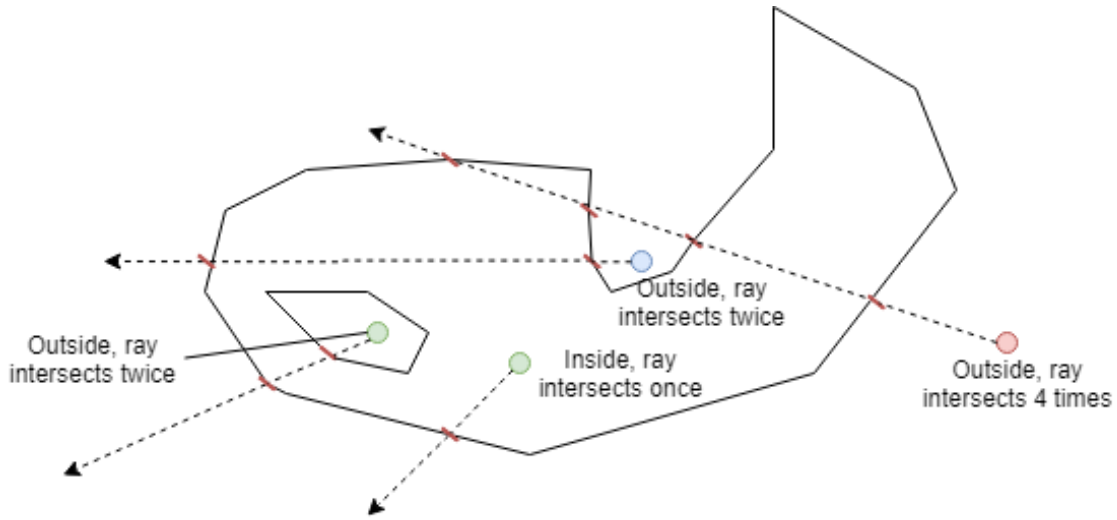


Figure 5.16. *Ray casting example.*

The system can contain multiple obstacles that have to be checked and that can increase evaluation time a lot. For the decreasing evaluation time and amounts of obstacles, we divide the system into sections. Every section contains only specific obstacles, which has

to be checked. The following figure illustrates warehouse splitting into sections.

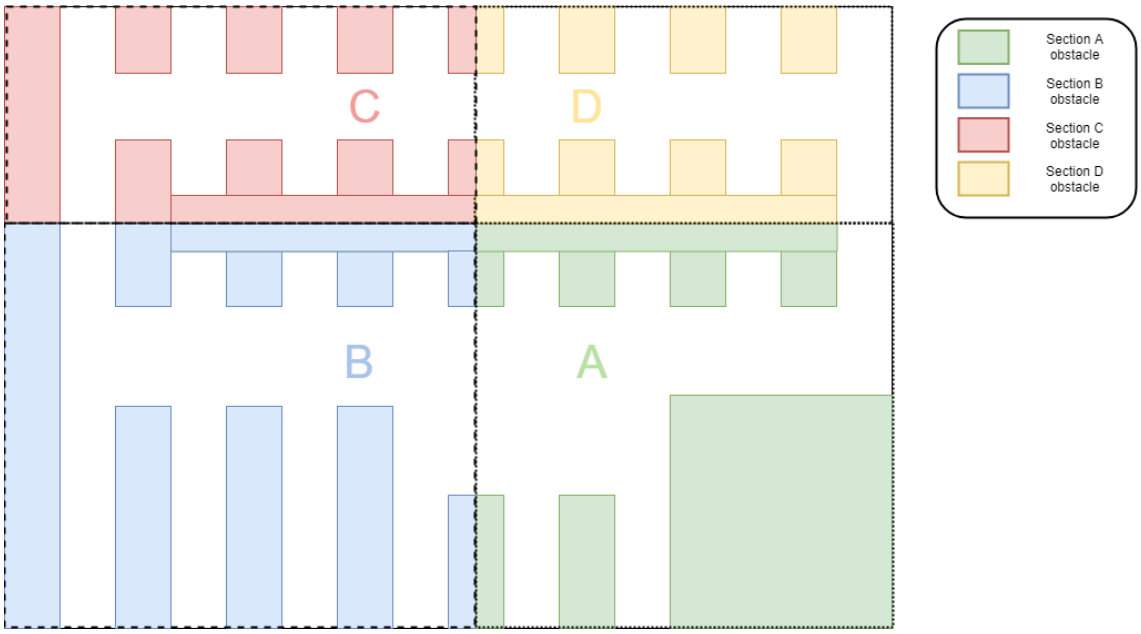


Figure 5.17. *Example of how warehouse can be splitted into sections.*

Now at the first, we need to specify the section where the object currently is and after that check collision only with the obstacles in that section. The section can be specified by using coordinates or by using a ray casting algorithm. [25]

5.3.3 Other AGV route detection to match real world system

In this algorithm, we haven't defined other AGV route detection or taken it to account as own part. We have left that out from the research, because we wanted to simplify the system when testing the genetic algorithm suitability for this kind of problem. This simplification doesn't affect the research results, because it is only one more restriction which will be added into fitness function. We still want to talk about problems and challenges of taking other AGV routes to account, also how they can be taken into account and also how it can be added into fitness function.

The first we will introduce three different scenarios of how taking the other routes can affect the optimization algorithm. The following figures contain an illustration of the three different cases.

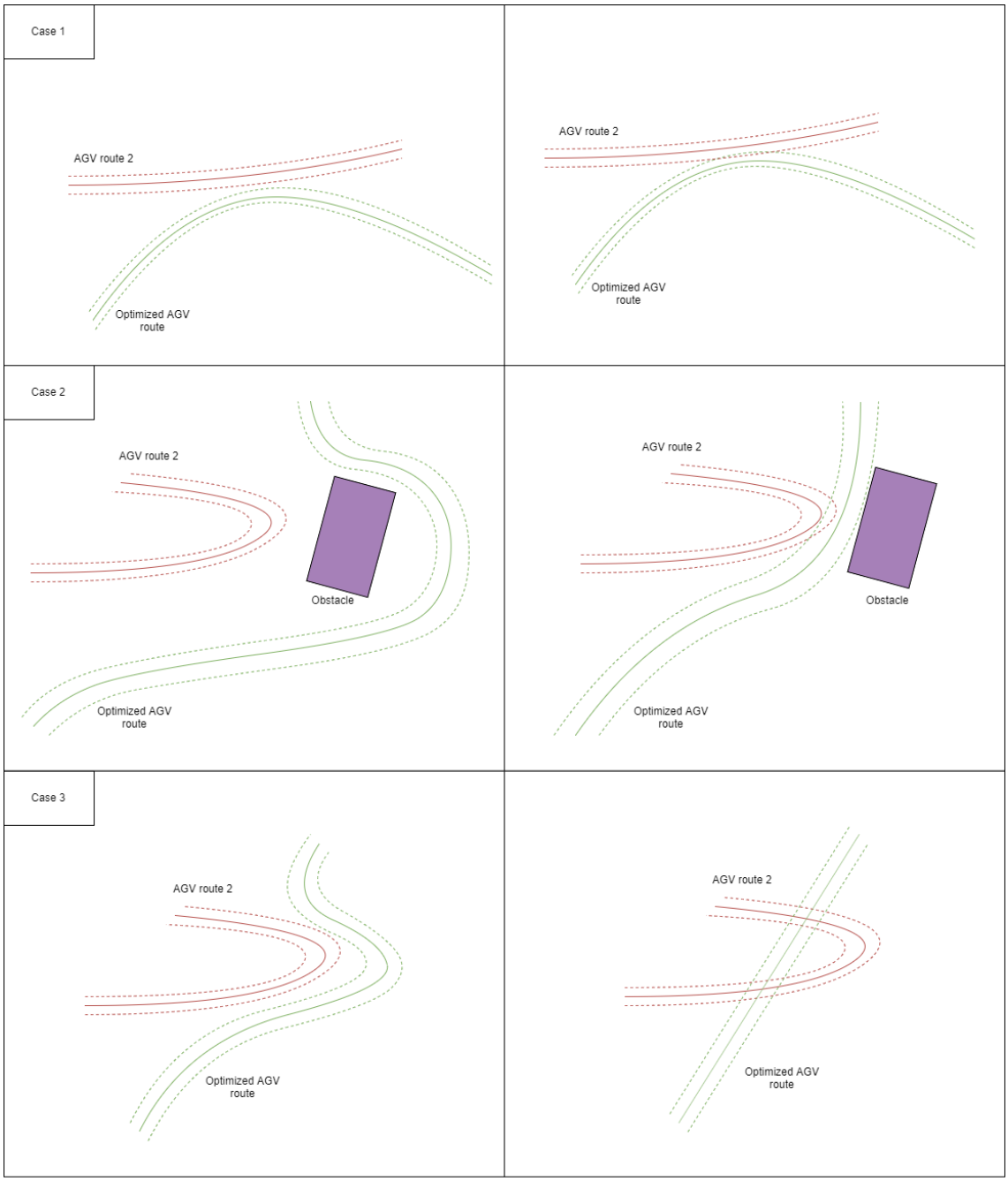


Figure 5.18. *Example of three different cases of how AGV routes collision could effect on optimization algorithm.*

In all cases, the left figure illustrates the "optimal" route when routes don't cross and the right figures illustrate the cases where routes cross. In real life and from the optimization point of view it doesn't matter how much our routes collide, but what matters is a number of times we crash to the other route. The size of the crash doesn't affect, because when the crash happens no matter how small it is, other AGV always has to stop to wait. What matters, is the number of times AGV has to stop to wait. In figure 5.18, in first case our route collides just a little. In this case, we want to avoid that collision and because of that, we have to penalize collision somehow at the fitness function. We want to penalize crashes that much that we can find the optimal route where crash won't happen.

In the second case, the route which won't crash has to evade the obstacle to get a clear route. This clear route is much slower than the route, which goes through the other AGV path. In this case, we have to decide if we want the algorithm to prefer drive time or the route "clearness". This same happens also in case 3. Preferring the clearness or the drive time is the case depended, because if the optimized route is some kind of maintenance route and it is used rarely, then it can go through the other path. If the optimized route is some kind of the main route, which has much traffic, then we should prefer the route clearness.

Preferring, either way, it will always affect the penalization function. If we want to prefer route clearness, then we should pick a little bit higher penalization for the route and if we want to prefer drive time, then we should pick a little bit smaller penalization. We want that penalization is dependent on the number of collision times. Let's mark collision times with c and "preferring" parameter be h, where $h \in [0, 2]$. If you want to prefer the route clearness, then h should be > 1 and respectively if you want to prefer time, then h should be $0 < h < 1$. Because routes have different drive times we want to scale penalization to match the routes time scale. For that we can get good time scale by dividing the length of the route by the half of the maximum velocity of the truck times ten. At the first dividing the length of the spline with the half of the maximum velocity gives one approximation for the drive time of the route. Then by dividing that with 10 gives the proper time scale for us. Because if the drive time would be $100s$, then by dividing that by 10 we would get $10s$, which is good time scale for the penalization in that case. For the penalization function, we can now just select

$$r(c) = \frac{l_{route}}{(v_{max}/2) \cdot 10} hc \tag{5.12}$$

$$= \frac{l_{route}}{v_{max} \cdot 5} hc \tag{5.13}$$

So now the full fitness function would be

$$f = t_{drive} + \sum_{j=1}^{n} (p(\frac{||\phi_j||}{\phi_{max}}) + g_j) + \frac{l_{route}}{v_{max} \cdot 5} hc. \tag{5.14}$$

For calculating the times that AGV routes collide we can use the same kind of method than for obstacle collision detection. We define all the other routes to be polygons, which are formed from their safe area borders or the sweeping polygons and then calculating the collision times the same way than in obstacle collision detection. We should divide the calculated number by two because always the second collision is detected when collision ends if the starting point is not colliding to the other AGV route already.

5.3.4 Changing the safe area to match sweep polygons

As we told earlier, in this research we used safe area creation for collision detection. This safe area defined to be such big that whole AGV is all the time inside the safe area. Safe

area creation was made such that its' border distance to the middle of route was constant at the all points. In real world this is not the case. When AGV drives the path its' corner points are in different distance from middle line in different points of route. This means that our safe area in current implementation is unduly big and because of that route could be optimized more in some cases. For solving this we would need to change our safe area creation so that it takes account AGV sweeping. The following figure has illustration of the current safe area creation and improved safe area creation which takes account a AGV sweeping.

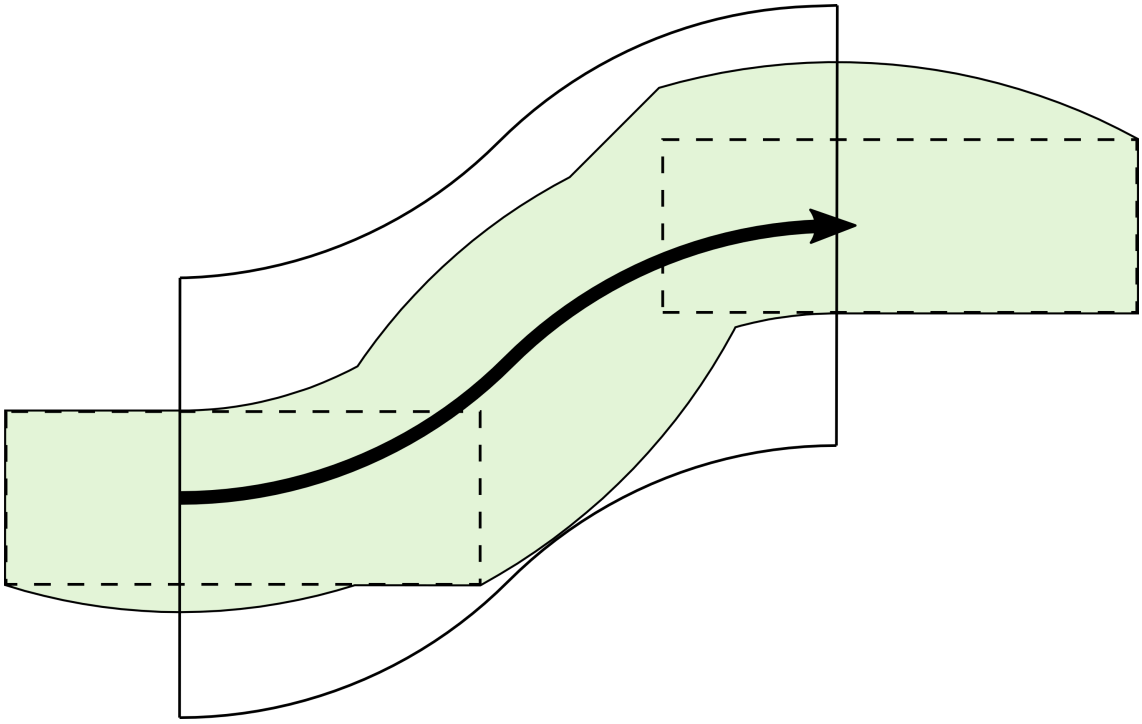


Figure 5.19. *Current safe area creation compared to improved safe area creation, which takes account AGV sweeping.*

As we can see from the figure, created safe area with improved creation process is much smaller in few places. In the figure dashed rectangle describes the AGV, green area is AGV's sweeping polygon and then the outter polygon with solid border is the current safe area creation implementation.

5.3.5 Algorithm usage for finding the fastest route between two given points

By slight modifications to the algorithm, it could be used for searching the shortest path between two given points. The first we would need an algorithm for creating the spline between the start and endpoint. This can be done easily by specifying the start and end point and then start and end angle. The spline can be then created by adding one control point by moving the start angle closer to the endpoint and then respectively moving from the endpoint by the end angle closer to start point and adding there a control point. After

that, we will add the required number of control points between previously added control points. The next figure will demonstrate how spline can be created.

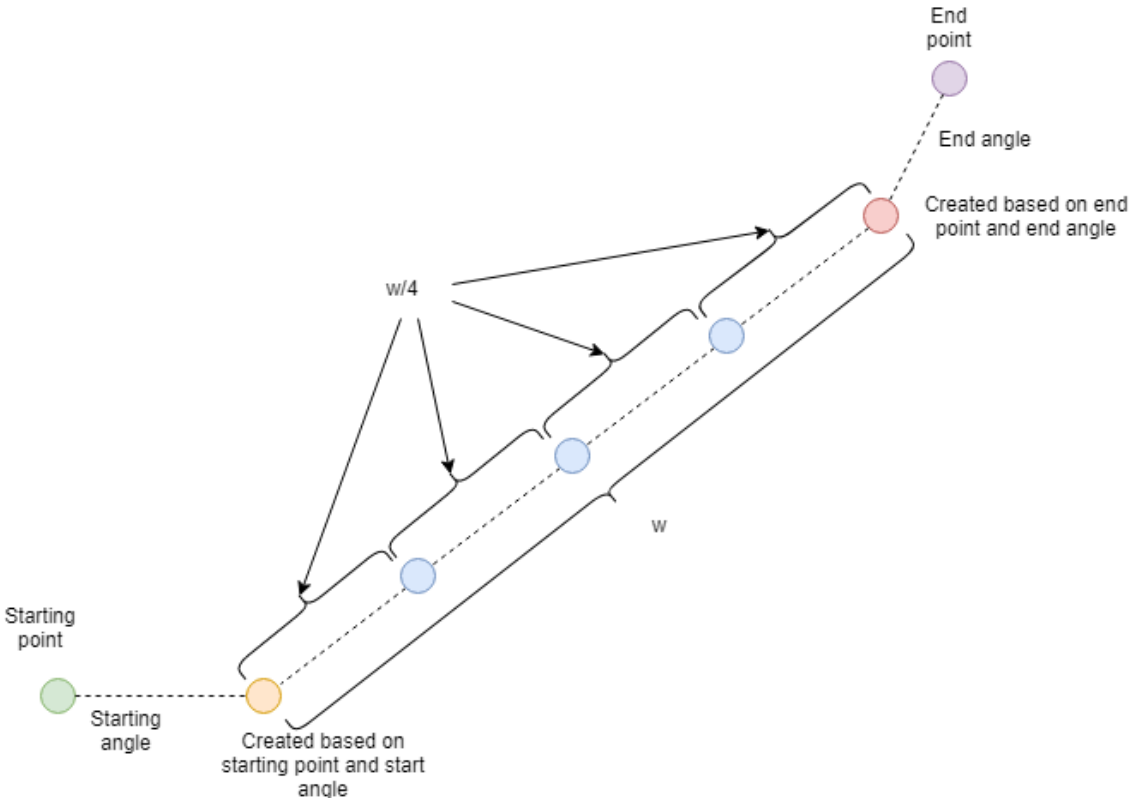


Figure 5.20. *Spline creation between two points, with starting angle and end angle.*

Now when we have spline between the start and endpoint, then we can use this in algorithm and try to find the optimal route.

6 OTHER APPROACHING PERSPECTIVES

In this chapter, we take a look into other approaching perspectives to the optimization process than a genetic algorithm. These approaches are not necessarily better than the genetic algorithm approach, but this purpose is to give more perspectives to the problem. The first we will take a look into the resilient backpropagation approach, which is a neural network approach to the problem.

6.1 Resilient backpropagation approach

Resilient backpropagation (RPROP) method is an artificial neural network learning algorithm that can be used for searching function minimum or maximum. RPROP method searches local minimums and maximums as the genetic algorithm does. In this method, we can't say that found minimum or maximum is the global minimum or maximum of the function. The RPROP method only considers a sign of the partial derivate of the weighting step and it won't take account of the size of the partial derivate. This means that this method only considers directions that won't change vehicle. Weight change is made by using "update-value". The RPROP algorithm has two steps in its learning process. The First step is to determine the sign of the "update-value" and the second step is calculating the "update-value" itself. In the first step sign of the "update-value" is determined by the sign of the partial derivate of weighting step as follows

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & \text{, if } \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & \text{, if } \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & \text{, else.} \end{cases} \tag{6.1}$$

In the previous equation, $\frac{\partial E}{\partial w_{ij}}(t)$ is the summed gradient information over all patterns of the pattern set. [8][24]

The Second step in the learning process is determining the "update-value". Value is calculated as follows:

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \cdot \Delta_{ij}(t-1) & \text{, if } \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^- \cdot \Delta_{ij}(t-1) & \text{, if } \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1) & \text{, else.} \end{cases} \tag{6.2}$$

In the previous equation, η^- is decreasing ratio of the "update-value" and η^+ is the growth ratio of the "update-value". For the η^- and η^+ must hold that $0 < \eta^- < 1 < \eta^+$. From the previous equation, we can see that if the sign of the update-value changes we move a little bit back because then we have moved over the local maximum or minimum. If the sign of the update-value stays the same, then the product $\frac{\partial E}{\partial w_ij}(t-1) \cdot \frac{\partial E}{\partial w_ij}(t)$ stays always positive. When sign stays the same, then we are moving towards a minimum or maximum depending on the current sign. If the sign is negative, then we are moving towards the local minimum and if the sign is positive then we are moving towards the local maximum. [8][24]

In most of the cases, decrease ratio η^- and growth ratio η^+ are set to fixed values before the algorithm. Now if the product of partial derivatives is negative, then we have moved over local maximum or minimum. This means that previous update-value has changed too much. We don't know how much we have gone over the maximum or minimum so we have to estimate it. In most cases halving the previous update-value is a good estimate. So we select $\eta^- = 0,5$. We have to also select the growth ratio η^+. The growth ratio has to be big enough to allow fast enough growth of update-value in small growth areas. On the other hand, we can't select too big η^+, because then the change of the sign occurs more often. It is commonly known and experiment many times that a 20% growth rate gives very good results in many cases. Slight change to that value doesn't affect much to results, but if we choose a much smaller growth rate, then growth is very slow and the optimal solution is reached much slower. On the other hand, if we choose a much bigger value, then sign change occurs too often and that slows down optimal solution finding. So let's select the growth rate to be 20%, so $\eta^+ = 1,20$. [24]

For this specific system, the RPROP method can be applied by optimizing finding an optimal solution for each system parameter one by one. After optimal is found, path is updated and we move to the next parameter. If we don't find better value for the parameter, then we won't update the path, we just move on to the next parameter. Each parameter is optimized by using the previously described RPROP method. For checking if we get a better solution, we compare fitness function current values for each parameter. [12]

6.2 Is RPROP better than GA?

The genetic algorithm has its weaknesses. At least in this implementation genetic algorithm seemed to be a little slow algorithm for the optimization, which little bit limits the possible applications for this algorithm. Although this algorithm can be improved many ways and it can get faster. General genetic algorithms weakness is the big reiteration times, for example, let's say we have the 10 sized population and for each generation, we create 5 new candidates. Now if our algorithm has to run 40 generations before it finds the optimal solution, then we have to create $9 + 5 * 39 = 204$ new candidates and evaluate the same number of candidates. 10 sized population is quite small and in many

cases, it is not enough. When the population grows, also our number of new candidates increases a lot. This means that our crossover and evaluation functions are executed many times. To get an efficient genetic algorithm we should develop fast crossover and evaluation functions.

The resilient backpropagation method works a little bit differently. In this method, we evaluate candidates the same way than in a genetic algorithm, but in this method, we don't create big populations or create new candidates during the algorithm. This means that we ran evaluation only to the one route which is much less than in genetic algorithm. This method finds the optimal solution quite different way than the genetic algorithm, in this method every property is optimized differently. This means that we go through all the control points in order and optimize every one of them. After the one control point is optimized, then we move into the next control point and so on to the last control point. [12] To get this method efficient we need to develop a fast optimization process for all properties.

The biggest problem of the RPROP method is finding the optimal solution. Because this method only considers the derivate sign. This method doesn't find the optimal solutions which are after the local maximum. The genetic algorithm tends to local maximum but by the mutation, it is possible to find an even better solution outside that local maximum. Another bad side of the RPROP is the optimizing the properties one by one, because as we know in splines control points always affect the current, previous and the next section of the spline. This means that when we optimize one control point, we affect into previous and next part also. RPROP method still can be used for route optimization, but it suits the best for the small optimizations and rarely it finds the best possible solution for the route.

7 SUMMARY

Our goal was to research the genetic algorithm suitability for the optimizing the AGV route. At first, we introduced splines and a few of its' properties. For this problem we used B-splines and for constructing them we used the Cox de Boor recursion algorithm [3]. For calculating the spline length, we introduced two different methods, the first one was multiple linear segments length approximation method and the other was the length of the parametrized curve. In the implementation, we used the multiple linear segments length approximation method, because it is much simpler to implement. We also introduced the B-spline derivation and curvature of the spline, which was used in the optimization algorithm.

We tested the algorithm with a few different test scenarios. The first we tested vehicle velocity evaluation after that we tested vehicles' collision detection and the genetic algorithms' population initialization. Finally, we tested full algorithm functionality. In vehicle velocity evaluation tests we tested velocity evaluation in five different routes with three different settings. Evaluation worked correctly and start velocity, end velocity, maximum centripetal, maximum acceleration, maximum deceleration and maximum velocity affected to the velocity graph correctly.

Collision detection was tested with four different routes and three different settings. Only settings that effected collision detection were AGV width and safe area width which were changed in different settings. In collision detection tests we tested that collision is detected correctly when AGV safe area collides to the obstacle. From the tests, we saw that the algorithm detected all the collisions where the safe area border cut the obstacle edge. We also noticed that there are a few cases where collisions would be not detected. One is the case where the whole obstacle is inside the AGV safe area. Collision detection is based on searching the cut points between safe area edge and obstacle edges and in this case they don't have any cut points. Collision detection also doesn't notice the case where the whole route is inside the obstacle. This collision detection worked for this research well, when we took into account all the limitations of the collision detection. This collision detection is not good enough for real-world usage and because of that we introduced the ray casting method which could solve these problems for real-world usage.

Population initialization was tested with four different routes. In this test, we tested that the population is initialized uniformly in both directions and also checked that the created routes seem reasonable. In all test cases, we got a reasonable population and routes were created uniformly distributed. We also measured population initialization times and

we got an approximation for the population initialization time, which depends on population size. For the population initialization we got $t_{init} = 0.051N + 0.048$, where N is population size.

The overall algorithm was tested in three different systems. The first system was a simple system, where there weren't any obstacles and the optimal solution should be the straight line. We ran optimization for every route 100 times with 10 different settings. We changed the population size from 20 to 60 and the mutation probability from 0.1 to 0.5. In the first test, almost a straight line was found in almost every case. Algorithm execution time varied from 26.67s to 123s and the average number of execution rounds varied from 37 to 45. In all cases, fitness values standard deviation was below 0,0085, which is quite good. In the second route, we had a little bit more complicated route and the system had also two obstacles. In this test population size changing and the mutation probability changing affected much more to optimization problem than in the first test case. In this execution time varied from 54,30s to 216,33s and respectively average execution rounds varied from 66 to 79. In this test, the standard deviation of fitness, values stayed under 0.12 and the found best fitness values average varied only 0,1. In the third case, execution times varied from 97,95s to 394,57s and in almost all the test cases average execution rounds were 100 or near it. This implicated that we should have raised the maximum number of rounds to get better routes. The standard deviation of fitness values varied from 0,5 to 2,45 but in most cases, standard deviation stayed under the 0,81. The found best fitness value average varied only 1,2.

As a result of testing, we get that the genetic algorithm can be used for optimizing the AGV routes, but the optimization algorithm is not the fast algorithm. This could limit the usage possibilities of the algorithm. There are still multiple ways to improve and speed up the algorithm, but the nature of the genetic algorithm is that it is never a very fast algorithm, because of its way to find the optimal solution. In complex cases growing the population will improve the results of the algorithm, but it will raise the execution time of the algorithm. So picking the correct population size for the problem can decrease execution time without impairing the found best solution too much.

REFERENCES

[1] F. Amato, F. Basile, C. Carbone and P. Chiacchio. An approach to control automated warehouse systems. English. *Control Engineering Practice* 13.10 (2005), 1223–1241. DOI: 10.1016/j.conengprac.2004.10.017.

[2] K.-E. Biebler and M. Wodny. *Splines and compartment models: an introduction*. English. New York: Springer, 2013. ISBN: 9789814522236.

[3] C. D. Boor. *A practical guide to splines*. English. Rev. Vol. 27; 27. New York: Springer, 2001. ISBN: 0387953663.

[4] T. L. Bui, P. T. Doan, S. S. Park, H. K. Kim and S. B. Kim. AGV Trajectory Control Based on Laser Sensor Navigation. English. *International Journal of Science and Engineering* 4.1 (2012), 16–20. DOI: 10.12777/ijse.4.1.16-20.

[5] P.-L. Buono. *Advanced calculus: differential calculus and Stokes' theorem*. English. Berlin: Walter de Gruyter GmbH, KG, 2016. ISBN: 9783110429114.

[6] P. J. Cameron. *Introduction to algebra*. English. 2nd; 2. New York; Oxford: Oxford University Press, 2008; 2007. ISBN: 1435633687.

[7] C. Canuto and A. Tabacco. *Mathematical Analysis I*. English. Milan: Springer Milan, 2008. ISBN: 8847008751. DOI: 10.1007/978-88-470-0876-2.

[8] A. Christofer, C. Kusuma, V. Pribadi and W. Budiharto. The Notation Scanner Systems Using Resilient Backpropagation Method. English. *Procedia Computer Science* 59 (2015), 98–105. DOI: 10.1016/j.procs.2015.07.342.

[9] Y. Falcone and C. Sánchez. *Runtime Verification : 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*. English. Vol. 10012. Cham: Springer International Publishing AG, 2016. ISBN: 3319469819. DOI: 10.1007/978-3-319-46982-9.

[10] K. Fortune. A local positioning system for AGV navigation. English. PhD thesis. 2016. ISBN: 1339-692260.

[11] T. Geveci. *Advanced calculus: using differential functions of several variables*. English. New York, [New York] (222 East 46th Street, New York, NY 10017): Momentum Press, 2016; 2015. ISBN: 9781606508787.

[12] M. Gloderer and A. Hertle. Spline-based Trajectory Optimization for Autonomous Vehicles with Ackerman drive. (), 1–6.

[13] E. Isaacson and H. B. Keller. *Analysis of numerical methods*. English. New York: John Wiley and Sons, 1966. ISBN: 9780471428657.

[14] Y.-B. Jia. Curvature. (Oct. 2019). URL: http://web.cs.iastate.edu/~cs577/handouts/curvature.pdf.

[15] O. Kounchev. *Chapter 11 - Appendix on Chebyshev splines*. ID: 273805. 2001. DOI: https://doi.org/10.1016/B978-012422490-2/50012-X". URL: http://www.sciencedirect.com/science/article/pii/B978012422490250012X.

[16] Y. K. Liu, X. Q. Wang, S. Z. Bao, M. Gomboši and B. Žalik. An algorithm for polygon clipping, and for determining polygon intersections and unions. English. *Computers and Geosciences* 33.5 (2007), 589–598. DOI: 10.1016/j.cageo.2006.08.008.

[17] M. Mansfield and C. O'Sullivan. *Understanding physics*. English. Second. Chichester, West Sussex: Wiley, 2011. ISBN: 1118437829.

[18] R. Oy. *Rocla AGV*. 27.8. 2019. URL: https://www.rocla-agv.com/.

[19] G. M. Phillips and P. J. Taylor. *Theory and applications of numerical analysis*. English. 2nd. London; San Diego: Academic Press, 1996. ISBN: 0125535600.

[20] L. Piegl and W. Tiller. *NURBS book*. English. Second Edition. Berlin/Heidelberg: Springer Berlin Heidelberg, Jan. 1997. ISBN: 3540615458. DOI: 10.1007/978-3-642-59223-2.

[21] A. Pressley. *Elementary Differential Geometry*. English. London: Springer, 2010. ISBN: 184882890X.

[22] X. Qian. Topology optimization in B-spline space. English. *Computer Methods in Applied Mechanics and Engineering* 265 (2013), 15–35. DOI: 10.1016/j.cma.2013.06.001.

[23] C. R. Reeves, J. E. Rowe and P. (Firm). *Genetic algorithms: principles and perspectives : a guide to GA theory*. English. Vol. ORCS 20.; ORCS 20. Boston: Kluwer Academic Publishers, 2003.

[24] M. Riedmiller. Rprop - Description and Implementation Details. (Jan. 1994), 1–2.

[25] T. Schroeder. Collision detection using ray casting. English. *Game Developer* 8.8 (2001).

[26] L. L. Schumaker. *Spline functions: basic theory*. English. New York: Wiley, 1981. ISBN: 0471764752.

[27] S. Shirali and H. L. Vasudeva. *Metric spaces*. English. London: Springer, 2006. ISBN: 1852339225. DOI: 10.1007/1-84628-244-6.

[28] S. N. Sivanandam, S. N. Deepa and I. Books24x7. *Introduction to genetic algorithms*. English. 4. Berlin: Springer, 2008; 2007. ISBN: 9783540731894. DOI: 10.1007/978-3-540-73190-0.

[29] A. H. Wright. The Exact Schema Theorem. English. (2011).

A POPULATION INITIALIZATION ALGORITHM

```
1  def initialize_population():
2      population = [self.starting_point]
3      for i in range(self.population_size-1):
4          cp_vector = []
5          cps = self.starting_point.get_cps()
6
7          # calculate distance between the first and last control point
8          m_p_d = distance(cps[0],cps[-1])/4
9
10         # Use the first two control points for the new candidate
11         cp_vector.append(cps[0])
12         cp_vector.append(cps[1])
13
14         orig_cps = self.starting_point.get_cps()
15         # Generate inner control points
16         for i in range(1,len(orig_cps)-3):
17             next_cp = cps[i+2]
18             prev_cp = cp_vector[-1]
19
20             # Check that reference line is not parallel to x-axis or y-axis
21             if cps[-1][0]-cps[0][0] != 0
22                 and cps[-1][1]-cps[0][1] != 0:
23                 k = (cps[-1][1]-cps[0][1])/(cps[-1][0]-cps[0][0])
24
25                 perp_dist = random.uniform(-1 * m_p_d, m_p_d)
26                 rand_ac = self.get_random_accidental()
27                 perp_x_div= math.sqrt(1 + (-1 * (k ** -1))**2)
28                 perp_x = rand_ac*perp_dist/perp_x_div+prev_cp[0]
29                 x_change = perp_x - prev_cp[0]
30                 perp_y = -1 * (k**-1) * x_change + prev_cp[1]
31                 n_cp_x = cps[i+2][0]
32                 n_cp_y = cps[i+2][1]
33                 max_x_den = k*perp_x+k**(-1)*n_cp_x-perp_y+n_cp_y
34                 max_x = max_x_den/(k + k**(-1))
```

```
35                  max_y = k * (max_x - perp_x) + perp_y
36
37                  dist = distance([perp_x, perp_y], [max_x, max_y])
38                  line_distance = random.uniform(0,dist)
39                  plus_x_div = math.sqrt(1 + k**2)
40                  plus_x = line_distance/plus_x_div + perp_x
41                  minus_x_div = math.sqrt(1 + k**2)
42                  minus_x = -1*line_distance/minus_x_div + perp_x
43                  line_x = plus_x
44
45                  dist_to_minus_x = abs(cps[-1][0]-minus_x)
46                  dist_to_plus_x = abs(cps[-1][0]-plus_x)
47                  if dist_to_minus_x <= dist_to_plus_x:
48                      line_x = minus_x
49
50                  line_y = k*(line_x - perp_x) + perp_y
51              elif cps[-1][0]-cps[0][0] == 0:
52                  # reference line is parallel to y-axis
53                  perp_dist = random.uniform(-1 * m_p_d, m_p_d)
54                  max_dist = abs(next_cp[1]-prev_cp[1])
55                  deltay = random.uniform(0,max_dist)
56                  line_x = prev_cp[0] + perp_dist
57                  line_y = prev_cp[1] + deltay
58              else:
59                  # reference line is parallel to x-axis
60                  perp_dist = random.uniform(-1 * m_p_d, m_p_d)
61                  max_dist = abs(next_cp[0]-prev_cp[0])
62                  deltax = random.uniform(0,max_dist)
63                  line_x = prev_cp[0] + deltax
64                  line_y = prev_cp[1] + perp_dist
65
66              cp = [line_x, line_y]
67              cp_vector.append(cp)
68          cp_vector.append(cps[len(cps)-2])
69          cp_vector.append(cps[len(cps)-1])
70          candidate = SplineCandidate(cp_vector, 3, n);
71          population.append(candidate)
72      return population
```

Listing A.1. *Spline population initialization functionality*

B CROSSOVER ALGORITHM

```
1  def crossover_children():
2      parents = select_random_parents(good_parents, bad_parents)
3      parent1 = parents[0]
4      parent2 = parents[1]
5      parent1_cps = parent1.get_cps()
6      parent2_cps = parent2.get_cps()
7      child_cps = []
8      for i in  range(2,len(parent1_cps)-2):
9          case = random.randint(0,2)
10         if case == 0:
11             # Use parent 1 control point
12             child_cps.append(parent1_cps[i])
13         elif case == 1:
14             # Use parents control points average
15             p1_x = parent1_cps[i][0]
16             p2_x = parent2_cps[i][0]
17             p1_y = parent1_cps[i][1]
18             p2_y = parent2_cps[i][1]
19             child_cps.append([(p1_x + p2_x)/2,(p1_y + p2_y)/2])
20         else:
21             # Use parent 2 control point
22             child_cps.append(parent2_cps[i])
23
24     child_cps = self.mutate_children_cps(child_cps)
25
26     child_cps.insert(0,parent1_cps[0])
27     child_cps.insert(1,parent1_cps[1])
28     child_cps.append(parent1_cps[len(parent1_cps)-2])
29     child_cps.append(parent1_cps[len(parent1_cps)-1])
30     return SplineCandidate(child_cps, 3, n)
31
32 def by_x(elem):
33     return elem[0]
```

Listing B.1. *Children crossover functionality*

C MUTATION ALGORITHM

```
1  def mutate_children_cps(child_cps):
2      if random.uniform(0,1) < self.mutation_p:
3          mutation_amount = random.randint(2,len(child_cps)-4)
4          avail_indexes = list(range(2,len(child_cps)-2))
5          for i in range(mutation_amount):
6              mutated_ind = random.randint(0,len(avail_indexes)-1)
7              mut_cp = mutable_indexes[mutated_index]
8              prev_cp = child_cps[mut_cp - 1]
9              next_cp = child_cps[mut_cp + 1]
10             mutation_range = distance(prev_cp, next_cp)/2
11             dist =  random.uniform(-mutation_range,mutation_range)
12             if  next_cp[1]-prev_cp[1] != 0
13                 and next_cp[0]-prev_cp[0] != 0:
14                 dy = (next_cp[1]-prev_cp[1])
15                 dx = (next_cp[0]-prev_cp[0])
16                 k = -1*(dy/dx)** -1
17                 cur_x = child_cps[mut_cp][0]
18                 cur_y = child_cps[mut_cp][1]
19                 deltax = (dist)/(math.sqrt(1 + k**2)) + cur_x
20                 deltay = k*(deltax - cur_x) + cur_y
21                 child_cps[mut_cp] = [deltax, deltay]
22
23             elif next_cp[0]-prev_cp[0] == 0:
24                 old_x = child_cps[mut_cp][0]
25                 child_cps[mut_cp][0] = old_x + dist
26             else:
27                 old_y = child_cps[mut_cp][1]
28                 child_cps[mut_cp][1] = old_y + dist
29             mutable_indexes.remove(mut_cp)
30
31         return child_cps
```

Listing C.1. *Children mutating functionality*

D FITNESS VALUE CALCULATION ALGORITHM

```
1  def p(c):
2      return exp(20*(c-0.9))
3
4  def phi(j):
5      return arctan(l_vehicle * k(j))
6
7  def g_j(j):
8      if not_allowed_area.Intersection(safe_area_rectangle) != {}:
9          return p(1000)
10     else:
11         return 0
12
13 def calculate_fitness_value(candidate):
14     n = 100
15     t_drive = calculate_drive_time(n)
16     penalty = 0.0
17     for i in range(n):
18         penalty_streeting = p(norm(phi(i)/phi_max))
19         penalty = penalty + penalty_streeting + g_j(j)
20
21     return t_drive + penalty
```

Listing D.1. *Fitness value calculation functionality*